Utrecht University

Dutch Ministry of Defence

Master Thesis

Solving a Ride-Sharing Problem for Dutch Ministry of Defence

Author:
Samuel F. van Rees

University supervisor:
Dr. J.A. (Han) Hoogeveen
Second reader:
Dr. ir. J.M. (Marjan) van den Akker
External supervisor:
Lkol. Stefan van Dijk

Computing Science

August, 2023

Universiteit Utrecht

Abstract

In this master thesis, we investigate a ride-sharing problem within the Dutch Ministry of Defence, characterized by multiple heterogeneous vehicles and depots, strict time constraints, consideration of user preferences, limitations on vehicle capacity, and flexible roles of drivers and riders. With a fleet of thousands of pool vehicles spread across multiple pool locations, optimizing vehicle usage becomes crucial due to emissions concerns and limited vehicle availability.

One possibility to achieve this is by introducing ride-sharing, wherein individuals are permitted to take a brief detour for the purpose of dropping off and picking up others. We propose a static simulated annealing algorithm to generate schedules that facilitate this ride-sharing and apply it in an iterated local search. Besides, we introduce the concept of decoupling vehicles from their original pool locations, resulting in the possibility of returning vehicles to different pool locations than their point of origin. The multi-objective function aims to minimize the number of non-served users, the ratio of the total distance travelled to that without ride-sharing, the number of non-empty vehicles, and the deviation from the desirable occupation of vehicles at pool locations.

Comparing our algorithm to the current manual approach, we observe significant improvements: increased number of planned users and reduced mileage through ride-sharing, while utilizing fewer vehicles and maintaining minimal deviation from the desired vehicle occupation at pool locations. Furthermore, combining ride-sharing and vehicle decoupling results in superior solutions, outperforming either approach alone. Secondly, we show the successful incorporation of single rides, which are rides from one pool location to another pool location without a return, and show that these increase the degree of ride-sharing. Lastly, our work presents a robustness analysis which demonstrates that rides can be effectively added to the generated schedules in a dynamic context. Moreover, although cancellations and delays cause a notable number of routes to become infeasible, a substantial portion remains feasible.

Acknowledgements

Although completing this thesis marks a high point in my academic journey and has mostly been a source of joy, the road to its completion was long and not without its challenges. I would like to express my gratitude to the individuals who supported and encouraged me along the way.

I want to start by thanking my university supervisor, Han. When I took my first course of the master's program with you as a teacher, I was impressed by your knowledge, teaching style, and enthusiasm. Despite being fully aware of the complex problems in the field and your high standards, I decided to challenge myself and undertake my thesis under your supervision. I did not regret this decision a moment. From the start of the process, you provided enthusiastic supervision, offered clear guidance, and posed insightful questions that guided me towards new insights. Your detailed comments on my draft submissions have truly polished this thesis into a well-structured work. Working with you has been a pleasure.

I also want to extend my gratitude to Stefan, my external supervisor at the Ministry of Defence. As Lieutenant Colonel and head of the unit responsible for the pool vehicles and -locations, you welcomed me with open arms into your department. Your patient explanations of the current planning system and your vision on its potential extensions were incredibly valuable. Our shared vision led to a productive and warm collaboration. Additionally, I would like to thank Merlijn, Colonel of the Transportation and Traffic Organization, who facilitated my entrance to the Ministry. Your enthusiasm and motivational attitude during our discussions about potential research directions and how my skills could contribute were incredibly valuable.

Furthermore, I would like to thank my friends and fellow students, Kasper, Olav, and Nick, who made my master's journey a truly enjoyable one. Your positive attitudes and determination created a healthy competitive environment where we pushed each other forward, supported one another, and most importantly, shared many enjoyable moments.

Lastly, I want to thank my other friends, housemates, girlfriend and family for their unconditional love, patience, and understanding. There were times I struggled to balance my thesis with other aspects of my life, and I sometimes lost sight of what truly mattered. Your understanding during those times and your unwavering support mean the world to me.

Contents

Abstract . . . iii
Acknowledgements . . . v
List of Definitions . . . x

1 Introduction **1**
1.1 Research Objective . . . 1
1.2 Shared Mobility Service . . . 2
1.3 Ride-sharing Variants . . . 4
1.4 Related Problems . . . 5
1.5 Contributions and Significance . . . 5
1.6 Outline . . . 6

2 Literature Review **7**
2.1 Related Problems in Literature . . . 7
2.2 Solution Methods . . . 9
2.2.1 Preprocessing Stage . . . 9
2.2.2 Matching Stage . . . 10

3 Problem Description **14**
3.1 Current Situation . . . 14
3.1.1 Depots . . . 14
3.1.2 Fleet . . . 14
3.1.3 Rides . . . 15
3.1.4 Users and Reservation . . . 15
3.1.5 Planning . . . 16
3.2 New Situation . . . 17
3.2.1 Depots . . . 17
3.2.2 Fleet . . . 17
3.2.3 Rides . . . 17
3.2.4 Users and Reservations . . . 18
3.2.5 Planning . . . 19

4 Objective and Constraints **20**
4.1 Objective . . . 20
4.2 Constraints . . . 22
4.2.1 Routing Constraints . . . 22
4.2.2 Vehicle Occupation Constraints . . . 23
4.2.3 Time Constraints . . . 23
4.2.4 Depot Constraints . . . 25

5 Data **26**
5.1 Pool Locations . . . 26
5.2 Requests . . . 27
5.2.1 Data Description . . . 28
5.2.2 Data Preparation . . . 29
5.2.3 Data Analysis . . . 33
5.3 Distance Matrix . . . 38
5.3.1 Filling frequent entries . . . 38
5.3.2 Filling remaining entries . . . 38
5.4 Recap on Data Preparation and Distance Matrix . . . 42

6 Solution Method **43**
6.1 Assumptions and Simplifications . . . 43
6.2 Representations and Implementations . . . 44
6.2.1 Graph . . . 44
6.2.2 Routes . . . 47
6.3 Preprocessing . . . 47
6.4 Initial Solution Heuristic . . . 48
6.4.1 Initialization . . . 50
6.4.2 Order of Placement . . . 50
6.4.3 Function Explanations . . . 50
6.4.4 Placement . . . 52
6.4.5 Constraint Checks . . . 53
6.5 Local Search . . . 54
6.5.1 Neighbourhood Operators . . . 56
6.5.2 Simple Constraint Checks . . . 59
6.5.3 Updates and More Complex Constraint Checks . . . 61

7 Methodology **63**
7.1 Three Comparative Algorithms . . . 63
7.1.1 Naive Approach . . . 64
7.1.2 Unbound Vehicles without Ride-sharing . . . 65

7.1.3 Ride-sharing with Bound Vehicles . . . 65
7.1.4 Returns with Bounds on Different Days . . . 67
7.2 Reducing Number of Vehicles . . . 68
7.3 Objective Hyperparameters . . . 70
7.4 Operator Hyperparameters . . . 73
7.5 SA Hyperparameters . . . 77
7.6 ILS Hyperparameters . . . 79

8 Experiments **80**
8.1 Experiment 1: Comparing four Approaches . . . 80
8.2 Experiment 2: Incorporating Single Rides . . . 83
8.3 Experiment 3: Robustness Analysis . . . 84
8.3.1 Adding Rides . . . 84
8.3.2 Removing Rides . . . 85
8.3.3 Delays . . . 85

9 Results **87**
9.1 Results Experiment 1: Comparing four Approaches . . . 87
9.2 Results Experiment 2: Incorporating Single Rides . . . 93
9.3 Results Experiment 3: Robustness Analysis . . . 95
9.3.1 Results Adding Rides . . . 95
9.3.2 Results Removing Rides . . . 96
9.3.3 Results Delays . . . 96
9.4 Conclusion of Results . . . 98

10 Discussion **99**
10.1 Limitations . . . 99
10.2 Implications . . . 102
10.3 Future work . . . 103

11 Conclusion **106**

A Experiment 1 Tables **108**

B Experiment 2 Tables **117**

C Experiment 2 Additional Figures **122**

List of Definitions

Driver and rider: A driver is defined as the one who drives the vehicle, while a rider is defined as one who is a passenger to the vehicle.

Node and arc: A ride request consists of two events: a pickup and a dropoff. Both can be represented as nodes in a graph. In this graph, an arc is established from node A to node B if it is potentially feasible, within the given time windows, for a vehicle to handle event A first, and event B subsequently.

Outbound and inbound rides: Returns have an outbound and inbound ride. In the return with locations A and B, the outbound ride is $A \rightarrow B$, while the inbound ride is $B \rightarrow A$. So these are respectively the outgoing ride from the starting point to the destination, and the ingoing ride from the destination to the end point - which corresponds to the starting point.

Mandatory and non-mandatory return rides: Outbound rides of non-mandatory returns are going to a pool location, and the inbound is not guaranteed to be scheduled if the outbound is. On the other hand, mandatory returns have a guaranteed inbound if the outbound is scheduled, as the outbounds are going to non-pool locations and the vehicles can not be left there.

Single ride: A single ride is a ride from one pool location to another pool location, which does not require a ride back. Single rides trivially only have an outbound ride.

Route: A route corresponds to the ride plan for a particular vehicle. Rides are placed in routes, and a route can fall into one of two types: *original* or *additional*. The *original* routes refer to the routes that form part of the final schedule, representing the actual routes to be driven by real vehicles. In contrast, the *additional* routes are scheduled for dummy vehicles and trips placed in these routes are not actually driven. Furthermore, it is possible for a route to be empty. If an original route is empty, it means the corresponding vehicle does not have any rides planned, and is thus unused.

Event time: The event time is the actual time an event, i.e. a pickup or dropoff, occurs at. Event times are scheduled as early as possible.

Chapter 1

Introduction

The *Defensie Verkeers- en Vervoersorganisatie* (DVVO) is, as a logistics service provider, responsible for all Dutch Ministry of Defence transport around the world. *Centraal wagenpark beheer* (CWB) is one of the four units of the DVVO. This unit possesses a few thousand passenger cars, encompassing petrol, hybrid, and diesel vehicles. A number of these vehicles are linked to individuals, but most of them are so-called pool vehicles. Ministry of Defence employees can use these pool vehicles for work-related appointments. In an app, they request a car at one of 54 pool locations. These pool locations are on military terrain, such as a military base. Employees are assigned a car from the pool location. They return it to the same pool location after their appointment. Defence's plans state that, as part of sustainability efforts, these cars must be replaced by electric cars or hydrogen cars by 2030 at the latest. The efficient utilization of cars has become an increasingly important issue in this context. A significant proportion of requests for car usage go unfulfilled due to a lack of available vehicles. In an effort to address this problem, one potential solution approach is the implementation of ride-sharing. This allows multiple individuals to share their ride, increasing overall utilization of the cars. Additionally, decoupling cars from their pool locations may further enhance the efficiency of car usage.

1.1 Research Objective

This thesis aims to investigate the potential of a simulated annealing algorithm used in an iterated local search for the static one-to-many ride-sharing problem, which incorporates decoupling cars from pool locations. This decoupling leads to the possibility of single rides, which are rides from one pool location to another pool location without a return. The ride-sharing problem being addressed is characterized by multiple heterogeneous vehicles and depots, strict time constraints,

consideration of user preferences, limitations on vehicle capacity, and flexible roles of drivers and riders. A driver is defined as the one who drives the vehicle, while a rider is defined as one who is a passenger to the vehicle. The multi-objective function aims to minimize the number of non-served users, the ratio of the total distance travelled to that without ride-sharing, the number of non-empty vehicles, and the deviation from the desirable occupation of vehicles at pool locations. Furthermore, our work presents a robustness analysis to evaluate the applicability of the generated schedules within a dynamic context, which involves adding new rides, removing rides, and including delays.

1.2 Shared Mobility Service

In recent years, there has been a noticeable change in attitudes towards consumption, with increasing attention given to the environmental, social, and developmental effects of consumption patterns. People are increasingly interested in finding sustainable and socially responsible ways of consuming. An example of this is the emergence of the *sharing economy* [11]. The sharing economy refers to the peer-to-peer-based activity of obtaining, giving, or sharing access to goods and services, coordinated through community-based online services. This sharing economy is expected to reduce societal problems such as hyper-consumption, pollution, and poverty [29]. Therefore, it is a rapidly expanding field and commercially attractive, as shown by several corporate researches [56] [65]. A growing body of academic research is being conducted in recent years as well, also indicating it is an area of increasing interest and significance [16] [35]. Mobility services that are based on the sharing economy represent a crucial aspect of this economic model and serve as an ideal example of the impact of the sharing economy on traditional industries. There are different types of shared mobility such as bike-sharing, scooter-sharing, on-demand ride-sharing and car-sharing [6]. It is important to note that there is a lack of consensus in the terminology and definitions associated with the shared mobility types and methods. In Figure 1.1, an overview of shared mobility service types is given.

As can be seen in this figure, there are two kinds of shared mobility service types. One is ride-sharing, while the other is car-sharing. The difference between these, is that in car-sharing individuals or organizations have access to a vehicle on an as-needed basis, without the cost and commitment of owning one. Ride-sharing, on the other hand, refers to the practice of multiple people sharing a ride in a single vehicle to reach a common destination. Thus, ride-sharing is different from the other shared-use mobility systems, in that not only the vehicle is shared, but the ride as well [67]. There are several types of car-sharing as shown in Figure 1.1.

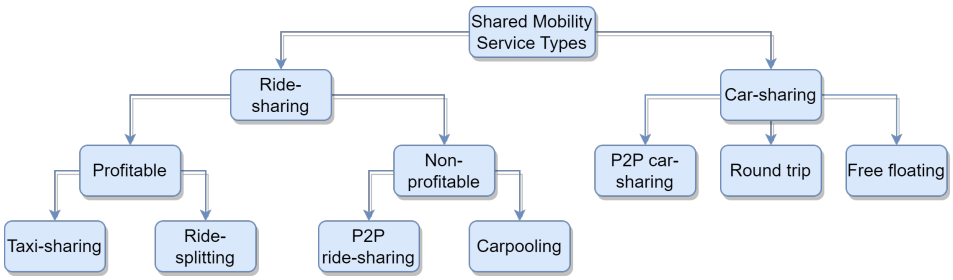

Figure 1.1: Shared Mobility Service Types

Peer-to-peer (P2P) car-sharing is a type of car-sharing service that utilizes web or mobile technologies to connect individuals who are willing to share their personal vehicles for short-term usage [25]. This form of car-sharing encourages the sharing of privately-owned vehicles among individuals and offers car owners the potential for generating additional income during periods of non-utilization [6]. Furthermore, Alarçin and Kirçova [6] (2020) make a distinction between round-trip based and free-floating car-sharing systems in car-sharing services. Round-trip based car-sharing refers to sharing services where cars are bound to depots or stations; cars are available at the stations of the service provider and brought back to the point where they are taken. Free-floating car-sharing refers to sharing services where cars are not bound to depots or stations; cars are parked in a suitable place within a designated area at the end of the ride, and trips are generally one-way [25].

Just like car-sharing, there are also several types of ride-sharing, as depicted in Figure 1.1. Ting, Lee, Pickl, and Seow [69] (2021) categorize ride-sharing systems into profitable and non-profitable. Taxi-sharing and ride-splitting are under the categories of profit-based ride-sharing. In taxi-sharing systems, a predetermined number of taxis are dispatched from a single depot or multiple depots to satisfy on-demand requests [39]. Sharing occurs if the taxi-driver allows other riders to get in the car while on a trip to the destination of the initial riders in the car [69]. In ride-splitting, riders are matched with similar origins and destination in real-time, and the cost is shared among riders [63]. The driver plays a role as a service provider, and the driver's intention is financial gain.

P2P ride-sharing and carpooling are classified as non-profitable ride-sharing systems. P2P ride-sharing is different from ride-splitting in that it is non-profitable, and the payment only covers the cost of the driver. A P2P ride-sharing system involves an online platform where users pre-register their trips, and a system operator matches riders and drivers and arranges their trips. Unlike traditional taxi-sharing systems, the drivers in P2P ride-sharing are also passengers who are willing to share their vehicle while carrying out their own trips. Thus, they generally have limited time windows and may not be available throughout the entire time horizon [67].

Traditional carpooling requires a long-term commitment among two or more people to travel together on recurring trips for a particular purpose, often for travelling to work [2]. Only travellers with a common origin to a common destination can be matched and served [69]. Therefore, carpooling is seen as a less flexible form of ride-sharing.

1.3 Ride-sharing Variants

All above examples of ride-sharing systems could be either dynamic or static. When decisions are made a priori, and all knowledge is known, the system is static. Conversely, in a dynamic system, the decision maker has the ability to modify existing plans in response to new information received during execution. Agatz, Erera, Savelsbergh, and Wang [2] (2012) describe dynamic ride-sharing systems as systems aiming to bring together travellers with similar itineraries and time schedules on a very short notice or even en route.

Furthermore, there are four basic ride-sharing variants, shown in Table 1.1, obtained from Agatz, Erera, Savelsbergh, and Wang [2] (2012). As shown in this table, drivers might take only single riders, or might take multiple riders. Similarly, riders might want a single driver, or may be willing to ride with multiple drivers and transfer from one to another en route. This leads to four variants: top left in Table 1.1 represents the *one-to-one* variant, top right represents *one-to-many*, bottom left depicts *many-to-one*, and bottom right represents the *many-to-many* variant.

	Single Rider	**Multiple Riders**
Single Driver	Matching of pairs of riders and drivers	Routing of drivers to pickup and deliver riders
Multiple Drivers	Routing of riders to transfer between drivers	Routing of riders and drivers

Table 1.1: Ride-sharing variants

The core of a ride-sharing system is a ride-matching problem that determines ride-sharing plans for users. One-to-one matching offers convenience for both drivers and riders, and is computationally efficient, making it a widely researched topic in the literature [67]. The other versions are computationally more complex. In the one-to-many variant, drivers can serve multiple passengers, but passengers cannot transfer between vehicles [8]. In the many-to-one scheme, drivers are limited to two stops; only one pickup and one dropoff, but passengers can transfer between vehicles [42]. The many-to-many variant is the most general, allowing for multiple vehicle

stops and passenger transfers [41]. The two variants, where multiple transfers are possible, are known as multi-hop [67].

Matching can occur under different system configurations, including fixed role, flexible role, and guaranteed ride back [3]. In fixed role matching, individuals register their trips with predefined roles as either riders or drivers. In flexible role matching, some individuals have the option to take on either role. A guaranteed ride back system ensures that riders have the ability to return home after their trip, making it an attractive option for commuters.

1.4 Related Problems

The planning problem we are solving in this thesis, briefly introduced in this section and further elaborated upon in Section 3, is a problem containing both aspects of car-sharing and ride-sharing. It has similarities with P2P ride-sharing in that participants register their trips ahead of time, a system operator matches riders and drivers and devises their itineraries, in that P2P ride-sharing system drivers are also customers who are willing to share their rides while completing their own trips, and that it is non-profitable. However, unlike typical P2P ride-sharing problems, in our case cars are not owned by drivers and riders, but are owned by a company. Therefore, the problem is also related to car-sharing.

Closely related to ride-sharing problems and our problem is the Dial-A-Ride Problem (DARP). In the DARP, individuals make requests for transportation from their starting locations to destinations. The service provider then uses their fleet of vehicles to arrange transportation for these requests, allowing multiple users with different requests to be transported in the same vehicle at the same time, thus sharing the service [34]. The DARP has many variants, such as using a single or multiple depots, using a homogeneous or heterogeneous fleet, and taking vehicle capacity, time windows, ride time, and route duration into consideration or not [34]. The most important difference between the DARP and our problem, is that there is no service provider who drives the customers. Instead, rides are conducted by employees themselves. In Section 2, we will look more closely at these related problems.

1.5 Contributions and Significance

This study aims to address a problem in the field of ride-sharing that has not been previously described in academic literature. The problem being studied is a unique combination of ride-sharing and car-sharing, where cars are owned by a company but shared between users. This research will contribute to the field

by providing new insights into the potential benefits and challenges of this type of service. Furthermore, the study extends the conventional ride-sharing problem by incorporating heterogeneous vehicles, multiple depots, strict time constraints, user preferences, and multi-objective optimization. This combination of extensions surpasses most studies in literature where problems are often less complex. Besides, other studies often deal with smaller-scale scenarios.

Moreover, this research will help the Ministry of Defence to make informed decisions about how to best develop and implement such a service. Besides, potential cost savings arise from two main aspects. Firstly, by automating the planning process, the need for manual labour in scheduling is minimized. Secondly, the efficient utilization of vehicles can lead to a reduced fleet size and a significant decrease in fuel consumption. This not only contributes to financial savings but also aligns with sustainability goals by reducing emissions.

1.6 Outline

The remainder of this thesis is outlined as follows. In Section 2, a literature review is given to provide an understanding of the current state of knowledge on related problems and solution methods. This will be followed by the problem description described in Section 3. In Section 4 we present the objective and constraints. After that, we elaborate on the used data in Section 5. In Section 6, we delve into the solution method, and Section 7 presents the methodology. Next, the experiments are explained in Section 8 and the corresponding results are presented in Section 9. In these experiments, we evaluate the performance of the developed algorithm, and analyse how it handles single rides, which are rides from one pool location to another pool location without a return. Additionally, we conduct a robustness analysis to evaluate the applicability of the generated schedules within a dynamic context. Lastly, in Section 10, we provide a discussion which elaborates on limitations, implications, and future research directions, while Section 11 concludes our work.

Chapter 2

Literature Review

The core of a ride-sharing system is a ride-matching problem that determines ride-sharing plans for users. The ride-matching problem is a generalization of the DARP [67]. The DARP generalizes a number of vehicle routing problems such as the Pickup and Delivery Vehicle Routing Problem (PDVRP) and the Vehicle Routing Problem with Time Windows (VRPTW) [20], which are variations of the Vehicle Routing Problem (VRP). Therefore, we will work top-bottom and start reviewing the VRP, and working our way down to the P2P ride-matching problem in Section 2.1. After this, we will look in Section 2.2 at solution methods to solve the DARP, P2P ride-matching, and some of their variants.

2.1 Related Problems in Literature

Vehicle Routing Problem In 1959, Dantzig and Ramser [23] introduced the Truck Dispatching Problem, modelling how a fleet of *homogeneous* trucks could serve the demand for oil of a number of gas stations from a central hub with minimum travelled distance. Clarke and Wright [17] (1964) generalized this problem: find the least-cost set of routes to serve a set of customers using a fleet of trucks with *varying* capacities. This became known as the Vehicle Routing Problem.

The classical VRP is static, and all data is obtained a priori. In real-world operations, data might be dynamic and not known in advance. Therefore, in the late 1970s, a more practical extension emerged: the Dynamic VRP (DVRP) [64] [71] [55]. For a long time, this area of research was not attractive, because of the lack of technological support, computational power, and business models. However, with new technology and techniques, innovative services, and business, new opportunities emerged. A number of papers on the DVRP have been published in the last decades, making it an active area of research [1] [61] [59]. The VRP typically deals with either pure delivery or pickup scenarios, where simultaneous presence of both deliveries

and pickups is not possible. However, in practical situations, it is common for customers to require both delivery and pickup services simultaneously. A variation of the VRP that deals with this is the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD) [44]. This, in combination with the transport of persons, leads to the core of ride-sharing problems.

Dial A Ride Problem The DARP generalizes a number of vehicle routing problems. In the DARP, all vehicles share the same origin and destination depot, and the loads to be transported are people [41]. These vehicles provide a shared service, allowing multiple users to be transported simultaneously. The objective is to minimize the cost of the routes while satisfying certain constraints. A common application of the DARP arises in door-to-door transportation services for the elderly and the disabled [20]. The DARP was initially designed to model paratransit systems [30], but has evolved over time to model the operations of ride-sourcing systems [18] [67].

What sets the DARP apart from most routing problems is the consideration from human perspective. When transporting passengers, reducing user inconvenience must be balanced against minimizing operating costs. Additionally, vehicle capacity is often a constraint in the DARP, whereas it is frequently redundant in other PDVRPs, particularly those related to the collection and delivery of letters and small parcels [20]. Passenger service quality may be measured, for example, in terms of the ratio of actual drive time and direct drive time, the waiting time, the number of stops while on board, and the difference between actual and desired delivery times [49]. These criteria may be treated as constraints or may be incorporated into the objective function [2]. Several surveys are provided on the DARP [34] [26].

P2P Ride-Matching Problem The P2P ride-matching problem is a generalization of the DARP [67]. The P2P ride-matching problem is distinct from the DARP, in that drivers in a P2P ride-sharing system are also considered customers. As a result, they possess unique origins, destinations, and time constraints, and are willing to share their rides while completing their own trips. This results in drivers having tight time windows and being unavailable for the entire time horizon [42]. Most studies on ride-sharing consider one, or a combination, of the following specific objectives when determining ride-share matches [2]: minimize system-wide vehicle miles, minimize the system-wide travel time, or maximize the number of served requests.

2.2 Solution Methods

Because our problem is closely related to P2P ride-matching and the DARP, we will review existing solutions to variations of these problems in the literature and look at both exact methods and heuristics and metaheuristics. Those problems are typically solved in two stages: preprocessing and matching. In the preprocessing stage, time windows are constructed, and the search space is limited by pruning. In the matching stage, the objective is to find an optimal match of requests.

2.2.1 Preprocessing Stage

Time Windows The discussed problems can be presented in a directed graph. Nodes present pickups and dropoffs of users. Nodes typically include time windows. Typically, an arc is present from node A to node B if it is potentially feasible, within the given time windows, for a vehicle to visit node B after it has visited node A. Arcs have values that represent the travel distance and travel time. Cordeau [18] (2006) provides a clear description of how time windows can be constructed. Cordeau notes that a user specifies either a desired arrival time at the destination, in the case of an outbound request, or a desired departure time from the origin, in the case of an inbound request. In the case of an outbound user, the time window at the origin node can be constructed using the arrival time window and in the case of an inbound user, the time window at the destination node can be constructed using the departure time window.

Pruning Often, initially these problems are presented as a complete directed graph. Because of time windows, pairing and ride time constraints, several arcs between nodes can be removed from the complete graph as their corresponding nodes cannot belong to feasible orders of node visiting. These arcs include arcs from dropoff nodes to their corresponding pickup nodes, and arcs between nodes which are infeasible regarding time windows and ride time limits [18].

Existing multi-hop algorithms mostly model the problem using time-expanded graphs (TEG) [42] [32]. In a TEG, nodes contain both location and time information representing the possible arrival times of a vehicle at each location. The nodes are connected by transfer and waiting edges to track the various routes a vehicle can take between locations and times [73]. Based on the key observation that the time constraints of the committed requests limit the area that a vehicle can reach, several authors prune by computing whether their reachable areas, which are presented as ellipses, cover the new request [73] [42].

Decomposition Furthermore, decomposing large-scale problems by dividing them into smaller subproblems, with for example clustering based on pickup and dropoff locations, may be beneficial. Subproblems which fall in the clusters are independent and can be solved in parallel. Ketabi, Alipour, and Helmy [38] (2018) developed a clustering algorithm based on a similarity score that accounts for both spatial and temporal similarities. Tafreshian and Masoud [66] (2020) proposed a polynomial-time graph partitioning technique that clusters trips based on a proximity measure and imposes uniformity between subproblem sizes to minimize solution time in a parallelized setting.

2.2.2 Matching Stage

Exact Methods for Ride-matching Problems One-to-one ride-matching problems, described previously in Section 1.3, can be represented in graphs, formulated as ILP's, and solved with known optimization methods. The maximum weighted bipartite matching problem can be used to determine how riders and drivers should be matched in the case of fixed roles, flexible roles, and guaranteed ride-back. There are many algorithms available for solving this problem with polynomial-time running time bounds [67].

The one-to-many ride-sharing problem, which involves assigning a single driver to multiple passengers, has been well-researched in the literature. Tamannaei and Irandoost [68] (2019) proposed a carpooling variant where the driver roles are assigned through optimization and not known in advance. They formulated this problem as a mixed-integer programming (MIP) and developed a branch-and-bound algorithm, as well as a beam search algorithm, to find near-optimal solutions for large-scale networks. Additionally, Armant and Brown [7] (2014) developed an MIP formulation with flexible roles, and solved it with constraints to avoid symmetrical solutions. In 2017, Masoud and Jayakrishnan [41] published a paper describing a MIP formulation for the many-to-many problem based on a time-expanded network. Besides, in 2017, they proposed an exact method for a dynamic many-to-one ride-sharing system [42].

Exact Methods for DARP Exact algorithms for DARPs are based mainly upon the concept of branch-and-bound; see for example Cordeau [18] (2006), Cortés, Matamala, and Contardo [21] (2010), and Hu and Chang [36] (2015). The development of exact methods for DARPs is focused on deterministic and static problems. The main reason is that exact methods may not be capable of providing timely solutions for dynamic DARPs. Important to note is that, to our knowledge, the largest instances that have been solved to optimality by exact methods for the basic DARP are up to 8 vehicles and 96 requests by Gschwind and Irnich [28] (2015).

For other variants, the largest instances are smaller because the variants are more complicated [34].

Neighbourhood Operators An important aspect of metaheuristics are neighbourhood operators. In literature, several operators are presented to solve the problem. Cordeau and Laporte [19] (2003) consider three types of simple moves, based on Nanry and Barnes [48] (2000). The first operation involves removing a pair of nodes from its current route and inserting it into a different one. The second involves exchanging two pairs of nodes between two separate paths, and the third consists of shifting a node within its current route. However, more advanced operators have been developed, such as the chain operator and the zero-split operator [50].

While a number of studies do not allow infeasible routes during the search, Cordeau and Laporte [19] (2003) allow solutions that violate time window and vehicle capacity constraints. Furthermore, as their problem includes ride time constraints and route duration constraints, the evaluation of the neighbourhood is complicated. They provide pseudocode to determine the arrival times in locations, the departure times, and so on, while minimizing violations in time windows, route durations, and ride times. The concept of *slack*, introduced by Savelsbergh [62] (1992), is used to achieve this. Slack indicates for each node how far the departure time of this node can be shifted forward in time without causing the route to become infeasible. This is an important component in scheduling problems, as it increases robustness in solutions, as for example shown by van Twist, van den Akker, and Hoogeveen [70] (2021).

Metaheuristics for Ride-matching Heuristic approaches are needed for large instances to provide timely, high-quality solutions for the ride-matching problem. Agatz, Erera, Savelsbergh, and Wang [4] (2011) represent the one-to-one ride-share problem with flexible roles using a general graph matching model and solve large instances with an iterative LP rounding heuristic to obtain a high-quality solution. Herbawi and Weber [33] (2012) formulated the dynamic one-to-many ride-matching problem as a Pickup and Delivery Problem with Time Windows (PDPTW), and they used a genetic algorithm (GA) to solve the problem for each time period, and an insertion heuristic to modify the solution in real time when a new request arrives. Furthermore, Herbawi and Weber [32] (2011) considered a variant of the multi-hop ride-sharing problem in which drivers have fixed routes and schedules. They formulated this problem as finding the shortest paths to earlier described TEGs with a multi-objective of minimizing costs, waiting times, and the number of transfers. They solved it using a multi-objective ant colony algorithm and combined it with a

local search yielding good results [31]. Ben Cheikh, Tahon, and Hammadi [9] (2017) propose a real-time multi-objective GA that uses fixed points along drivers' routes to enable transfers. Chen, Mes, Schutten, and Quint [15] (2019) solved a dynamic many-to-many ride-sharing system with a ride-back guarantee. For instances with more than 80 participants, they developed a greedy heuristic which obtains high-quality solutions.

Metaheuristics for DARP Since DARP is NP-hard, the focus of much research has been on developing efficient and effective heuristic techniques. Metaheuristics as Tabu Search (TS), Simulated Annealing (SA), Variable Neighbourhood Search (VNS), and Large Neighbourhood Search (LNS) have been widely applied to solve DARP-variants [34].

VNS is a metaheuristic that alternates neighbourhoods systematically in a local search to prevent being stuck in local optima by switching between multiple operators [45]. Parragh, Doerner, Hartl, and Gandibleux [51] (2009) proposed the first VNS for the DARP using an SA-acceptance criterion, where four different neighbourhood operators were used. This served as a foundation for other research on using VNS to solve the DARP, including solving more complex DARPs and using additional operators [24] [47] [46].

LNS involves removing requests from the current solution, and then trying to insert them in a better way. The changes that are made to the solution are relatively large. In LNS, typical destroy operators are random removal, worst removal, sequential removal, route removal and related removal, while common repair operators are random insertion, greedy insertion, k-regret insertion, most-constrained-first insertion and space-time-related insertion [45]. The Adaptive LNS (ALNS) is an extension of LNS that adaptively chooses among a number of insertion and removal heuristics, to intensify and diversify the search [53]. Ropke and Pisinger [60] (2006) developed an ALNS algorithm for the PDPTW, using simple and fast removal and insertion heuristics, and an SA-acceptance criterion. This method is effective and efficient, and has served as groundwork for subsequent studies on LNS for DARPs [27] [40]. LNS is beneficial for variants where transfers are possible, since operators can be developed which take these transfers into consideration [43] [13].

Cordeau and Laporte [19] presented in 2003 a TS-algorithm for the DARP. In this algorithm, infeasible solutions are temporarily accepted. This algorithm has been found to be effective and efficient, and many recent studies on TS for the DARP followed this work [24].

Although SA has not been used as frequently as other metaheuristic approaches to solve the DARP, it has been used with basic operators obtaining satisfactory

results [57] [74]. Braekers, Caris, and Janssens [12] (2014) proposed a highly efficient and effective variant of SA, called Deterministic Annealing (DA). In DA, worse solutions than the current one are accepted if the difference in the objective value is smaller than a gradually lowered threshold value. The consequence of this is that the search is more structured and controlled than SA, which involves randomness in accepting worse solutions. They used complex neighbourhood operators and a restart strategy when the search became stuck. This combination of methods provided excellent results for various DARP variants. Masmoudi, Hosny, Braekers, and Dammak [40] (2016) integrated a population-based component into DA and SA by developing a hybrid bee algorithm, which outperforms the results of Braekers, Caris, and Janssens [12] (2014) for the multi-depot DARP.

Insertion Heuristics Simple insertion methods have been developed to solve the standard DARP and its variations. These heuristics are often based on the idea of the greedy insertion heuristic introduced by Jaw, Odoni, Psaraftis, and Wilson [37] in 1986. In context of the DARP, a pickup and corresponding dropoff of a customer are inserted by considering all possible insertion points in existing routes and evaluating these with respect to the objective function. Insertion heuristics serve as a good starting point for further optimization, as demonstrated by Braekers, Caris, and Janssens [12] (2014). Furthermore, these construction heuristics are effective in providing fast solutions for dynamic DARPs, as demonstrated by Wong, Han, and Yuen [72] (2014). The reason for this is insertion heuristics can add new requests without requiring re-computation of the complete solution, as also shown by Coslovich, Pesenti, and Ukovich [22] (2006). Furthermore, van Twist, van den Akker, and Hoogeveen [70] (2021) combine an insertion heuristic with a small-scale local search, to handle new passengers with reduced mobility in existing schedules for assistance from airport employees.

Moreover, in context of adding new requests to existing solutions, a number of other methods have been proposed. For instance, Pouls, Meyer, and Glock [54] (2021) combine a dispatching algorithm with a local search. Riley, van Hentenryck, and Yuan [58] (2020) present a real-time dispatching solution for a ride-sharing service with a rolling horizon that utilizes a column-generation approach. A computational study shows that their approach scales very well in practice.

Chapter 3

Problem Description

We provide an overview of the current status, presenting the current state of the fleet and depots, the current planning process, and the user interaction with the system in Section 3.1. Subsequently, in Section 3.2, we discuss the changes resulting from the implementation of a planning algorithm that complies with the desires of the Ministry of Defence. Moreover, we will discuss the specific modifications within the system and the consequences for the users.

3.1 Current Situation

In this section, we will look at the situation with the current planning system in use. More specifically, we will look at the depots, fleet, rides, users, reservations, and planning.

3.1.1 Depots

There are 54 pool locations distributed across the Netherlands which are located on military terrain, such as military bases. These pool locations are shown in Figure 3.1. These pool locations are the depots from which vehicles depart and return to. Vehicles are linked to depots; a vehicle should always return to its depot. Thus, only returning trips are possible.

3.1.2 Fleet

According to our data, the Ministry of Defence currently has a heterogeneous fleet existing of approximately 2200 petrol, hybrid and diesel vehicles. The fleet consists of different types of vehicles: cars with a capacity of 2, 4, 5, or 8. Present vehicles are passenger cars, delivery vans with a small cargo bed, bigger vans with a large

Figure 3.1: Pool locations across the Netherlands

load capacity, and 5-door passenger cars with ample baggage space. In Section 5.1 more information on the number of vehicles per type in the fleet is given.

3.1.3 Rides

It is important to note that during the period when an employee has the car reserved, they have the freedom to utilize the vehicle as they see fit. While rides are monitored, the employee retains the flexibility to choose the route, arrival, and departure times to a particular location. The only restrictions are that vehicles always have to return to their pool location, that the car is accessed at most one hour after the start of the reservation, and that the car is returned before the end of the reservation slot. Thus, an employee has the car for a certain time slot - which can be multiple days, and can do whatever he likes in this time slot, as long as it is accountable and responsible. The cost of gas is covered by Defence. Rides are not shared in the current situation, meaning that two employees, who have similar departure and arrival times and locations, will be assigned separate cars.

3.1.4 Users and Reservation

Defence employees can use the pool vehicles for work-related appointments. In an app, they request a car at one of 54 pool locations. Employees are assigned a car from the pool location. In this app, the employee can specify among others the following:

- **Time window of reservation.** This indicates the specific time period during which the employee wants to reserve a vehicle.
- **Departure location.** Note that this has to be a pool location.
- **Arrival location.** Note that this does not have to be a pool location per se. Many requests have as arrival location for example Schiphol, although this is not a pool location. Since the car will always be returned to the departure location, this is not a problem.
- **Number of passengers.** Although ride-sharing is not implemented in the current planning system, employees can decide themselves to share a ride. If so, they can indicate with how many they do.
- **Type of car.** If users decide themselves to share their rides with colleagues or have lots of luggage, they can for example choose to specify they want a passenger van. If they travel alone, they can indicate they want a small passenger car.

More on the input of users is described in Section 5.2.1. Immediately after the reservation is made, the employee receives an email. In this email, the location of the car, the type of car, and the license plate are given. The employee accesses the car with a smart card.

3.1.5 Planning

Planners manually assign cars to employees. In a planning software, planners are notified when a new request comes in. Per pool location, they have an overview on which cars are used for which time slot and which cars are available. They look at the request specifications, and assign an available car. It could be the case that the required type of car is already occupied. In this case, an alternative type of car is given. If all cars at a depot are taken, an option for a car at a nearby depot is given. Normally, one or two days in advance, most requests have come in. They work with a *first come first serve* strategy. Employees, who hand in their request early, are more certain of a car. Cancellations and changes in requests are taken care of by the planners manually. They might call employees to report changes in the planning if needed.

3.2 New Situation

As described in Section 1.1, we will develop an algorithm for generating schedules, which incorporates ride-sharing and decouples cars from pool locations. These implementations change some of the aspects described in Section 3.1. In this section, these and the preferences of Defence that come with these modifications are described.

3.2.1 Depots

Depot locations do not change. However, as some pool locations are very closely located, these are bundled into one. More information on this is provided in Section 5.1. Furthermore, in the new situation, vehicles are no longer restricted to return to their designated depot. However, it is desirable that cars remain evenly distributed across pool locations to avoid clustering at a single location; it is not preferable for a pool location to have no vehicles available due to users only departing from it and not returning to it. Nevertheless, small variations in the end occupation are allowed. More on this is described in Section 4.1.

3.2.2 Fleet

Apart from the fact that the planning algorithm does not take into account the type of fuel the vehicle runs on, there are no changes occurring in the fleet. At each pool location, the actual types and quantities of vehicles are utilized. However, in the experiments conducted in this thesis, the number of vehicles is reduced. More details regarding this can be found in Section 7.2.

3.2.3 Rides

In the context of rides, many changes take place for employees. For oversight, the changes are listed and discussed below.

- **No vehicle possession.** Users no longer possess a vehicle for a specified period of time, which they can utilize as they see fit, but instead receive a schedule to which they must adhere. This may require them to extend their ride time, transport other individuals, or occasionally wait.
- **Flexible roles.** Roles are flexible, and it may be possible for individuals to shift from being a driver to a passenger and vice versa.
- **Returning at end of the day.** It is no longer possible to have the vehicle for multiple days. Each vehicle must be returned to one of the pool locations by the end of the day. As can be read in Section 5.2.3, a number of reservation

are made for multiple days. This decreases the efficiency and utilization of vehicles drastically. Therefore, vehicles should be returned to a pool location by the end of the day.

Although users surrender some autonomy and privilege, there is an increased efficiency in utilizing the vehicles and a reduced mileage because of these limitations. Besides, single trips to other pool locations are possible which ensures that employees do not necessarily have to return the car to the pool location from which they came, which can be advantageous for them.

As can be further read in Sections 4.2.1 and 5.2.2, there are different types of request available. The first type corresponds to the single ride, which involves a trip from one pool location to another pool location without a return. Additionally, there are returns, which can be classified as mandatory or non-mandatory returns. Returns have an outbound and inbound ride. In the return with locations A and B, the outbound ride is $A \rightarrow B$, while the inbound ride is $B \rightarrow A$. So these are respectively the outgoing ride from the starting point to the destination, and the ingoing ride from the destination to the end point - which corresponds to the starting point. Single rides trivially only have an outbound ride. Although a return's outbound and inbound are dependent of each other, as can be further read in Section 4.2, they could be seen as two single rides. Non-mandatory returns do not have a guaranteed ride back, as they are going to a pool location and the vehicle can be left there. On the other hand, mandatory returns have a guaranteed ride back as they are going to non-pool locations, and the vehicle needs to return to a pool location. Although it is not possible to reserve a vehicle multiple days, it is possible for users to plan their inbound and outbound on different days. More on this specific case is explained in Section 4.2.1.

3.2.4 Users and Reservations

The user base remains unchanged. However, the introduction of ride-sharing and decoupling vehicles from their pool location, brings several modifications for them. Initially, all employees hold the role of drivers; they may now also serve as riders, so roles are flexible. Additionally, the process of making reservations undergoes modifications, with employees now having other options to specify their preferences. Through the use of an app, an employee can specify the following details:

- **Request type.** As explained before, different request types are possible. It is possible to request a single ride, mandatory return, or non-mandatory return. Outbounds and inbounds of returns might occur on different days.
- **Departure and arrival location.** In single rides, there is only one departure and one arrival location, and both are pool locations. In return trips, there

are two departure and two arrival locations, one each for the outbound and inbound ride. It is important to note that the departure location of the outbound ride corresponds to the arrival location of the inbound ride, and vice versa. As previously explained, in a non-mandatory return, the arrival location of the outbound ride is a pool location, whereas in a mandatory return, it is not a pool location.

- **Maximum ride time.** An employee indicates its maximum ride time in its reservation. In the experiments in this thesis, we will account 1.2 times the actual ride time for this, as further explained in Section 5.2.2.
- **Departure and arrival time windows.** Time windows correspond to a user's nominal travel time and maximum travel time. More information on this can be found in Section 5.2.2. Time windows must be satisfied, as further explained in Section 4.2.3; cases where employees are too late for appointments because they had to drop off someone else, are not desirable.

3.2.5 Planning

In this thesis, we design a planning algorithm for the one-to-many ride-sharing. By considering requests that are known a few days in advance, the algorithm is able to solve the problem as a static problem. Nevertheless, in Sections 8.3 and 9.3 we will also evaluate the applicability of the generated schedules within a dynamic context, which involves adding new rides, removing rides, and including delays. In the following sections, we will delve further into this planning algorithm.

Chapter 4

Objective and Constraints

In the planning algorithm, two important aspects are the objective function and the constraints. The objective function evaluates solutions by assigning them a fitness score. This function comprises various components, making it multi-objective in nature. On the other hand, the constraints ensure that the found solutions are feasible. Both the objective and constraints are discussed in Sections 4.1 and 4.2, respectively. Before elaborating on the multi-objective function, it is crucial to discuss an essential aspect of the problem representation, which is further discussed in Section 6.2.2. A route corresponds to the ride plan for a particular vehicle. Requests are placed in routes, and a route can fall into one of two types: *original* or *additional*.

Original and Additional Routes: The *original* routes refer to the routes that form part of the final schedule, representing the actual routes to be driven by real vehicles. In contrast, the *additional* routes are scheduled for dummy vehicles and trips placed in these routes are not actually driven. Furthermore, it is possible for a route to be empty. If an original route is empty, it means the corresponding vehicle does not have any rides planned, and is thus unused.

4.1 Objective

The objective function comprises four components: the ratio of the total distance travelled to the total distance travelled without ride-sharing, the number of non-empty vehicles, the number of users that cannot be accommodated within existing vehicles, and the deviation between the actual and desirable numbers of vehicles remaining at the pool locations by the end of the day. All of these components should be minimized. Note that in the first two components, i.e. the distance component and non-empty route component, only original routes are considered as these are the routes actually driven. Also note that the number of users that

cannot be accommodated within existing vehicles is equal to the number of users that are placed in additional routes. Furthermore, minimizing the number of non-empty vehicles is not a direct goal in itself. However, incorporating this component is crucial in order to avoid requests that could have been accommodated serially in the same vehicle ending up being assigned to separate vehicles. Further details on this matter can be found in Section 7.3. Moreover, adhering to the requirements of Defence, small deviations in vehicle occupation at pool locations are considered insignificant, while larger deviations carry greater significance. To capture this behaviour, an exponential function is used to represent the deviation between the actual and desirable numbers of vehicles. By raising the exponential function to the power of the absolute difference between these values, the objective function is designed to place more emphasis on larger deviations compared to smaller ones. All components are multiplied by a hyperparameter, enabling the decision maker to assign weights to each component based on their relative importance. Consequently, the multi-objective function is formulated as follows:

$$\min \left(\sum_{i=1}^{I} \left[x_i \left(\alpha \frac{d_i}{v_i} + \beta z_i \right) + \gamma l_i \left(1 - x_i \right) \right] + \sum_{j=1}^{J} \sum_{k=1}^{K} \left[\delta e^{\,|o_{jk}|} \right] \right)$$

In this formulation, the hyperparameters and variables present the following:

α : Hyperparameter for total travel distance with ride-sharing ratio component.

β : Hyperparameter for the number of non-empty vehicles component.

γ : Hyperparameter for non-served users component.

δ : Hyperparameter for occupation deviation at pool locations component.

I : Number of routes (both original and additional).

J : Number of pool locations.

K : Number of vehicle types.

$$x_i : \begin{cases} 1, & \text{if route } i \text{ is original.} \\ 0, & \text{if route } i \text{ is additional.} \end{cases}$$

d_i : Actual travel distance of route i.

v_i : Travel distance of route i without ride-sharing.

$$z_i : \begin{cases} -1, & \text{if route } i \text{ is empty.} \\ 0, & \text{otherwise.} \end{cases}$$

l_i : Total number of users in all rides in route i.

o_{jk} : Difference between actual and desired occupation of type k at pool location j.

4.2 Constraints

To ensure feasibility, a number of constraints should be satisfied by all routes in the solution. As discussed before, a route corresponds to the ride plan for a particular vehicle. In the subsequent subsections, these constraints are listed and discussed. Note that all these constraints should be satisfied by both additional and original routes. The only exception is Constraint **1.**, which does not have to be satisfied in additional routes.

4.2.1 Routing Constraints

1. **Start and end point**
   All non-empty original routes should originate and end at a pool location. The departure pool location and arrival pool location of a vehicle should not necessarily be the same. As explained in Section 4.1, an even distribution of vehicles over the pool locations is maintained by the objective function. Note that this constraint does not have to be satisfied for additional routes. As these routes are not driven, it is not a problem if no vehicles are available at the departure time of the first user or if the vehicle is left at a non pool location.

2. **Exactly one route per corresponding pickup and dropoff**
   The pickup and corresponding dropoff must be assigned to exactly one and the same route. Note that this route could be either additional or original. This constraint ensures that each customer is placed in at most one vehicle, and that a pickup and dropoff can not be split between multiple routes.

3. **Pickup before dropoff**
   Pickup must occur before dropoff for each customer. This requirement ensures that ride plans are formed in a logical and sequential manner.

4. **Inbound and outbound rides**
   As can be further read in Section 5.2.2 and is already shortly described in Section 3.2.3, a distinction is made between single rides, non-mandatory returns, and mandatory returns. Returns are, as described in Section 3.2.3, further divided into an outbound ride and an inbound ride. In the case of non-mandatory returns, unlike the mandatory ones, it is possible for the outbound ride to be scheduled without the inbound ride. However, for both it is not permissible for the inbound ride to be scheduled without the outbound ride being scheduled, as the inbound ride cannot be executed independently. A summary of this information is presented in Table 4.1. Note that 'original'

Return Type	Outbound	Inbound	
		Present	Absent
Non-mandatory	Present	Feasible	Feasible
	Absent	Infeasible	Feasible
Mandatory	Present	Feasible	Infeasible
	Absent	Infeasible	Feasible

Table 4.1: Feasibility for present and absent in- and outbound rides in mandatory and non-mandatory returns

and 'additional' correspond to the terms 'present' and 'absent' in Table 4.1, respectively.

As described in Section 3.2.3, it is no longer possible for individuals to possess a vehicle for multiple days. Vehicles are returned to a pool location by the end of the day. However, inbound and outbound rides of returns can occur on different days. For returns with bounds on different days, specific arrangements must be specified. The outbound of a mandatory return, occurring on a day different from the inbound, will never be placed at the end of an original route. Such a placement would violate Constraint **1.**, as the car would end up at a non-pool location by the end of the day. Consequently, these users can only reach their destination by being dropped off in a ride-sharing arrangement. This means that the concept of a guaranteed ride back no longer applies in this situation, as there is no car stationed at this location to provide the user with a return trip on the day of the inbound. The only way to return is by sharing a ride with another user, which is not always guaranteed. Therefore, when a mandatory return has an outbound present, the inbound cannot be guaranteed. Thus, when the outbound and inbound of a mandatory return are requested on different days, the ride back, unlike the normal situation, is no longer assured.

5. Outbound before inbound

Outbound rides should always occur before the corresponding inbound ride.

4.2.2 Vehicle Occupation Constraints

6. Maximum occupation

The number of users in a vehicle must not exceed the maximum capacity of that vehicle, which is not constant but varies per vehicle type.

4.2.3 Time Constraints

7. Event time in time window

Each event, whether it is a pickup or dropoff, is assigned a time window within

which it can take place. The time windows are consistent with the nominal ride time from the pickup to the dropoff location and the maximum ride time. The maximum ride time is the time the user is willing to travel. More on these time windows and ride times can be read in Section 5.2.2. The actual time the event occurs at, is referred to as the *event time*. The event time for departure must fall within the departure time window, while the event time for arrival must fall within the arrival time window.

Earliest Possible Event Time All event times are scheduled as early as possible, aiming to increase slack and robustness, which is important when dealing with delays and changes in the schedule. More on this can be read in Sections 8.3 and 9.3. For the starting point in a route, the event time is set as early as possible within its time window. The event time of the next event is calculated by adding two components to the previous event time: the service time, which is always two as explained in Section 5.2.2, and the ride time of the previous event to the next event.

There are three possibilities regarding this newly calculated event time, as shown in Figure 4.1. The first possibility is that the event time occurs before the time window begins. In this case, the event time is dragged forward to the earliest possible moment of the time window. The second possibility is that the event time occurs within the time window. In this case, nothing happens as it is feasible. The third possibility is that the event time occurs later than the end of the time window, making it infeasible. By scheduling events as early as possible, we aim to make the solution more robust. By doing this, the effects of delays and real-time changes are minimized. More on this robustness is discussed in Sections 8.3 and 9.3.

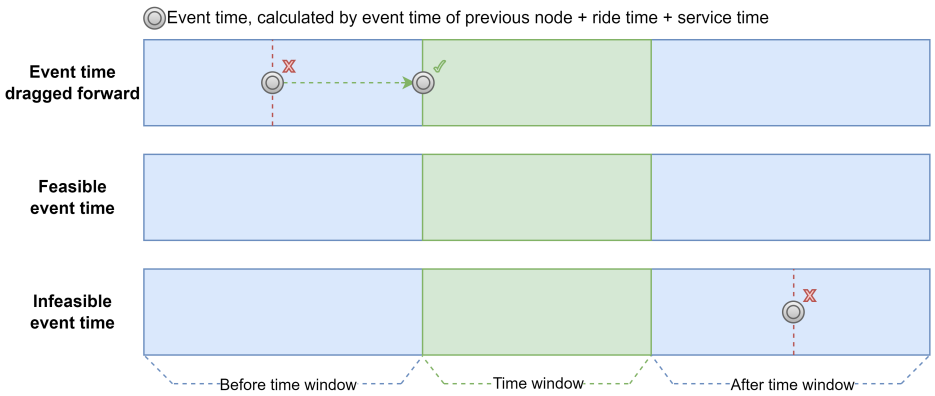

Figure 4.1: Calculating event times

8. Maximum ride time

In addition to the requirement of the event time falling within the time window, the difference in event time of the pickup and dropoff should not exceed the maximum ride time. More on the maximum ride time is elaborated in Section 5.2.2. As we will also observe in this section, it might be possible that Constraint **7.** is satisfied, while the maximum ride time is exceeded.

4.2.4 Depot Constraints

9. Availability of vehicles

As illustrated in Section 5.1, the pool locations are equipped with various types of vehicles. As explained earlier in Section 4, routes are associated with the ride plan of a specific vehicle. Therefore, each route corresponds to a particular vehicle type. A route is considered feasible only if, at the event time of the first pickup location on that route, i.e., the location from which the vehicle departs, there is still a vehicle of this specific type available. Further details on this aspect can be found in Section 6.5.3.

Chapter 5

Data

In this section, we provide an overview of the data available for the pool locations and requests. Specifically, we describe the data for the pool locations in Section 5.1 and the data for the requests in Section 5.2.1. Subsequently, we explain the data preprocessing techniques employed in Section 5.2.2 to make the data suitable for conducting experiments in Section 8. Following that, we present a concise data analysis of the request data in Section 5.2.3. In Section 5.3, we elaborate on the process of creating a distance matrix.

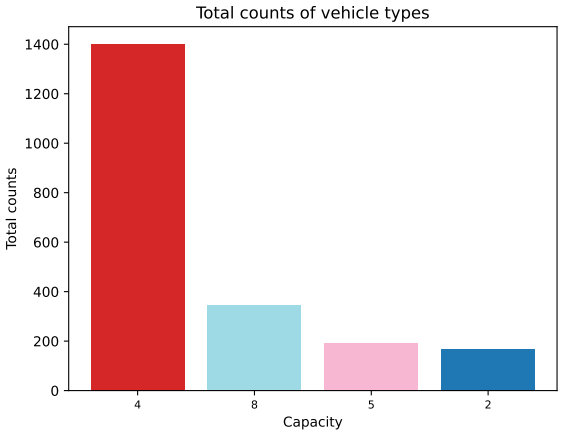

Figure 5.1: Total number of vehicles per type

5.1 Pool Locations

An important database used, provides an overview of the pool locations. In this Dutch-language *xlsx*-file, all pool locations are presented with their official abbreviations, their location, and the number of vehicles of each type per pool location. As

described earlier in Section 3.1.2, there are four different types of vehicles: vehicles with capacity 2, 4, 5 or 8. In Figure 5.1, the total number of vehicles per type is given. We can observe that the most occurring type is the one with a capacity of 4. Following that, the larger vehicles with a capacity of 8 are the next most common, followed by vehicles with capacities of 5 and 2.

Some pool locations are very closely located. In Den Helder, there are five different pool locations, in Darp two, in Gilze Rijen two, and in Doorn two. These very closely located pool locations are clustered and seen as one single location with their coordinates being the middle point of the others. In Figure 5.2, the number of vehicles per type per pool location is presented. As can be seen, some pool locations as the combined pools of Den Helder have up to 143 vehicles, while other pool locations have as few as six vehicles, as for instance Willem Lodewijk van Nassaukazerne, with abbreviation 'MARNE'.

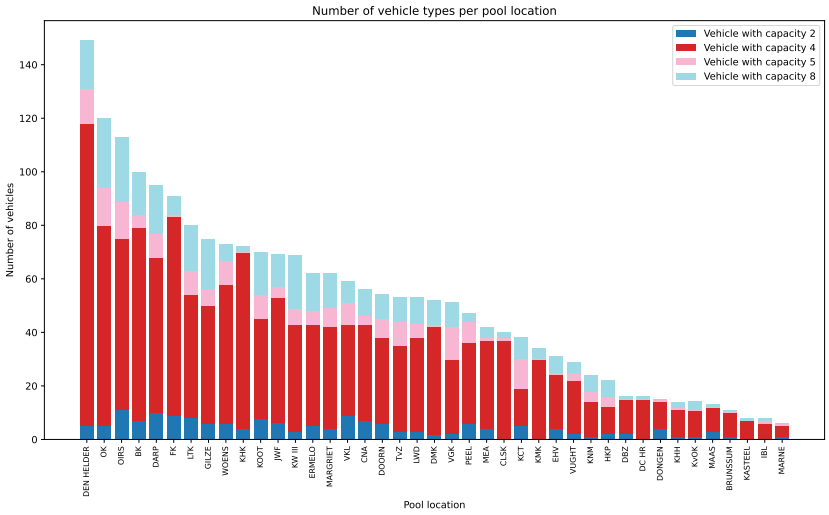

Figure 5.2: Number of vehicles per type per pool location

5.2 Requests

For developing and evaluating the planning algorithm in Section 8, which can be applied in the new situation described in Section 3.2, historic request data is used. We will describe this raw data in Section 5.2.1, and in Section 5.2.2 explain how we prepare it to be usable for the planning algorithm. In Section 5.2.3 we analyse the data.

5.2.1 Data Description

Request data is obtained from a file granted by Defence, containing all anonymized ride requests for the period from January 1, 2022, to January 31, 2022, including additional information about the nature of the requests. It was provided in a Dutch-language *xlsx*-file with 25,161 rows and 55 columns. Each row represents a single request for reserving a car, including both accepted and rejected requests. All requests in this database correspond to return rides, as described in Section 3.1.3. So for the return trip with outbound $A \rightarrow B$ and inbound $B \rightarrow A$, a single row is used, indicating the departure location as A and the arrival location as B. So the database provides the departure and arrival locations for the outbound ride. Columns we will use are listed below and are briefly described.

- **Request ID** is a unique combination of digits for representing the request.
- **Request date** represents the specific date and time, accurate to the minute, when a request is submitted.
- **Departure date** represents the specific date and time, accurate to the minute, when a reservation starts.
- **Return date** represents the specific date and time, accurate to the minute, when a reservation ends.
- **Request status** represents the status of the request. Examples of possible values are 'planned', 'in progress', 'cancelled by planner', 'cancelled by user', and 'vehicle not picked up'. A request is cancelled by a planner if for example no cars are available for that specific time slot.
- **Cancelled** is a column with binary values, representing whether the request is cancelled or not. This column corresponds with the *request status*-column.
- **Number of passengers** represents the number of passengers. As described earlier in Section 3.1.4, although ride-sharing is not implemented in the current planning system, employees can decide themselves to share a ride. If so, this column represents with how many they do.
- **Departure location** represents the location from which the vehicle is picked up and returned to. Note that this is always a pool location.
- **Arrival location** represents the destination of the user of the outbound ride. Note that this is not always a pool location.

5.2.2 Data Preparation

In this subsection, the steps taken in the data preparation process to make the data suitable for the experiments described in Section 8, which are consistent with the new situation as described in Section 3.2, are described. We will explain which unusable rows and columns are removed, how a string similarity algorithm is applied, how ride types are determined, and how time windows are formed.

Removal of Unusable Rows and Columns Several rows contained empty fields in the departure location column. These rows are deleted as they are unusable. Similarly, a number of rows have departure locations which are not pool locations. These corresponding rows are deleted as well. In Section 7.2 is explained how we compensate for these removed requests. Additionally, requests that are directly or indirectly cancelled by the user are also removed. This includes requests cancelled by the user themselves, requests cancelled due to the user's non-confirmation, or cases where the vehicles were not picked up. Furthermore, all columns are removed except for those described above, such as the column representing the reason for the request and the column indicating whether the ride is function-related or not.

String Similarity A main issue with the request dataset is that users manually type their destination, which is recorded in the *departure location*-column. As a result, there is an inconsistency in the destinations. For instance, the location 'Kromhoutkazerne' in Utrecht is indicated not only as 'Kromhoutkazerne' but also as 'Kromhoutkazerne Utrecht', 'Kromhout', 'KHK', or simply 'Utrecht'. To deal with this problem, two string similarity methods from the Python module *Fuzzywuzzy* are used; `fuzz.ratio` and `fuzz.partial_ratio`. Both methods calculate the edit distance between token orderings in the input strings using the `difflib.ratio` function. This function returns a similarity score between 0 and 1, where 1 represents identical sequences and 0 indicates no similarity. The first method, `fuzz.ratio`, directly calls `difflib.ratio` on the two input strings. On the other hand, `fuzz.partial_ratio` aims to handle partial string matches more effectively. It uses the shortest string and compares it against all substrings of the same length of the longer string, returning the highest score. The manually written destination is compared using these two methods against the actual names of all pool locations, the official abbreviations, and the places they are located in. Subsequently, the maximum score from these comparisons is chosen, and if it surpasses 0.82, which is a threshold determined through trial and error, the destination is assigned the name of the corresponding pool location. Furthermore, the remaining destinations are compared with each other, and clustered together and given the same name, if they achieve a score higher than 0.90 on either of the two methods,

again determined through trial and error.

After the majority of the locations have been automatically cleaned, the remaining portion is addressed manually. In the database is searched for destinations with values as 'Local', 'Hospital', 'Diverse', 'XXX', or 'Regional', as these destinations lack a specific destination. These requests are all deleted as they are unusable. Certain locations appeared more frequently, but could not be linked to a geographic location. Often, these were unofficial abbreviations that occur within the jargon of the personnel. An example of this is the use of 'ISK' instead of 'Generaal Winkelmankazerne'. These locations have been replaced with the official names of the corresponding pool location.

As explained, we substitute place names with a pool location if one is located in that place, so for example 'Utrecht' will be substituted by 'Kromhoutkazerne'. When the departure location is also 'Kromhoutkazerne', the departure and arrival locations become identical. This is a problem because, as described, the database only provides the departure and arrival locations for the outbound ride. When the locations are identical, the outbound of the return trip would appear as $A \to A$, and the inbound would also be $A \to A$. To address this issue, arrival locations in rows with the same locations in the specified columns are replaced with the place name where the pool locations are located. Thus, in the case of a departure location of 'Kromhoutkazerne' and an arrival location of 'Kromhoutkazerne', the arrival location is replaced with 'Utrecht'.

Ride Type Subsequently, a column called 'ride type' is added. Values in this column are 'mandatory return', 'non-mandatory return', or 'single ride'. As described in Section 3.2.3, a request is classified as 'mandatory return' when the ride is going to a non-pool location. This type of ride is mandatory because the vehicle needs to return; it cannot be left there. Rides going to pool locations are labelled as 'non-mandatory return'. Then, from the 'non-mandatory return' rides, a random 10% of the rides are selected and labelled as 'single ride.' This percentage was determined through discussions with Defence, as it is expected to represent the proportion of rides that would be one-way if given the option. Lastly, the rows labelled 'mandatory return' or 'non-mandatory return' are duplicated. The originals are assigned the value 'Out' in an added column called 'Bound', while the duplicates are assigned the value 'In' to represent respectively out- and inbound rides. In these duplicates, the departure location and arrival location are swapped, resulting in outbounds and inbounds of the form $A \to B$ and $B \to A$. Ultimately, after deleting unusable rows and duplicating and editing returns, for the month of January a total of 46,111 rows, corresponding to ride requests, are present in the database.

Time Windows Another issue with this dataset is the absence of time windows. To evaluate the algorithm for the new situation described in Section 3.2, these time windows need to be established. There should be departure and arrival time windows within which a user can depart and arrive, as specified by Constraint **7.**. These time windows need to be created based on the values in the columns 'Departure date' and 'Return date', as this is the only time-related data we have. To establish time windows for outbound rides, the departure date is utilized, while the arrival date is used for inbound rides. It was decided, again through discussion with the decision-maker, that departure time windows for outbounds and arrival time windows for inbounds should be 20 minutes for realistic results. Besides, it is discussed that service-time, such as getting in and out of vehicles and denoted as d_i at departure or arrival i, takes two minutes. The maximum travel time, L, is determined by increasing the nominal travel time, $t_{i,n+i}$ by 20%. The time windows for departure i and its arrival $n+i$ are established as follows:

- **Departure time window for outbound rides.** The time recorded in the *departure date*-column is considered the latest departure time, l_i, for the outbound ride. This is because it represents the time users have specified for their departure in their reservations. It cannot be later than this, as it may already be the deadline for users to leave and arrive on time. The earliest departure time, e_i, for the outbound ride is determined by subtracting 20 minutes from this time as shown in Figure 5.3.

- **Arrival time window for outbound rides.** The earliest arrival time for the outbound ride, e_{n+i}, is calculated by adding the nominal travel time and service-time to the earliest departure time. The latest arrival time, l_{n+i}, is determined by adding the maximum travel time and service-time to the latest departure time. By doing this, we have created an arrival time-window from the departure time-window, which perfectly fits the feasible arrival time window. If a person departs at the earliest moment possible and drives the nominal ride time, and accounts for the service-time, the user will get there at the earliest arrival time. Besides, if a user departs at the latest possible departure time and the trip takes as long as the maximum ride time, the user will arrive at the latest possible arrival.

- **Arrival time window for inbound rides.** For inbound rides, the value in the *return date*-column is used to establish time windows. The latest arrival time is set to this value since it represents the time when users must return the vehicle at the latest. The earliest arrival time for inbound rides is determined by subtracting 20 minutes from this time, as shown in Figure 5.4.

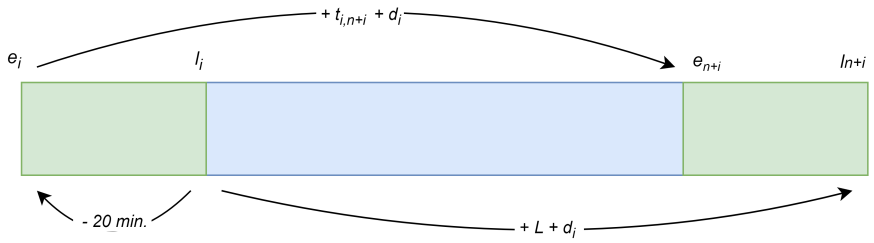

Figure 5.3: Setting time windows for outbound rides

- **Departure time window for inbound rides.** The earliest departure time for inbound rides is determined by subtracting the maximum ride time and service-time from the earliest arrival time. The latest departure time is calculated by subtracting the nominal ride time and service time from the latest arrival time.

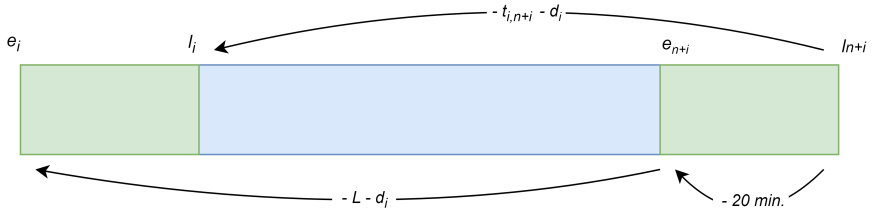

Figure 5.4: Setting time windows for inbound rides

If a user departs at the earliest possible time and the trip is performed with nominal ride time, the person will arrive before the arrival time window begins, as shown in Figure 5.5. Conversely, if a user departs at the latest possible departure time and the trip takes longer than the nominal ride time, the user will arrive later than the latest possible arrival, as also shown in Figure 5.5. This figure shows we have a bigger reachable arrival time window that does not perfectly match the smaller feasible arrival time window but includes all points of this feasible time window. Rides that arrive before the start of the arrival time window must wait until the arrival time window begins, and rides that arrive later than the arrival time window are declared infeasible. This concept is discussed in Section 4.2.3. It is worth noting that waiting-time provides more robust solutions that better handle cancellations and other changes. More on robustness can be read in Sections 8.3 and 9.3.

Note that for both inbound and outbound rides, it is possible for the maximum ride time to be exceeded even if the event times of pickup and dropoff fall within their corresponding time windows. This is the reason why satisfying solely Constraint **7.** is not sufficient, but Constraint **8.** must also be fulfilled.

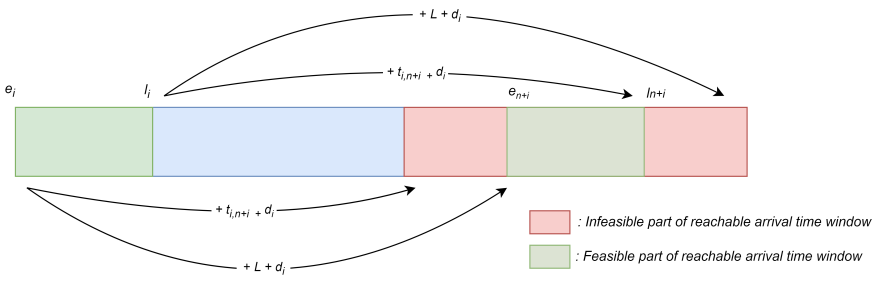

Figure 5.5: Feasible and infeasible part of reachable arrival time window, from the constructed departure time window

5.2.3 Data Analysis

To get insight in the edited request data, a visualization is provided in this subsection. We will look at the number of requests per day, reservation durations, departure and arrival locations, time windows, and maximum ride times.

Requests per Day As described in Section 5.2.2, the utilized request data corresponds to the period of January 2022. Figure 5.6 presents a bar plot depicting the number of requested departures per day for both outbounds and inbounds. It is evident that there are fewer requests during weekends compared to weekdays. Additionally, a noticeable decrease in requests is observed during the first week following New Year's Day, which is often a holiday week. The number of requests appears to increase in the second week and stabilizes in weeks 3 and 4, providing a more accurate representation of the daily request count.

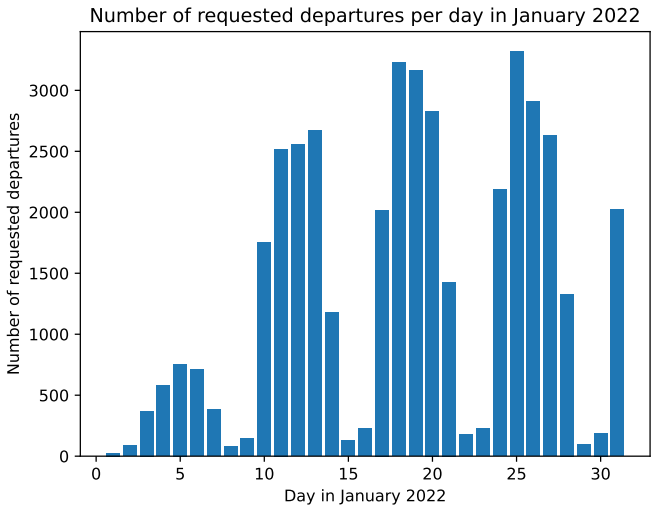

Figure 5.6: Number of requested departures per day in January 2022

Reservation Duration As previously discussed in Section 3.2.3, it is possible for the inbound and outbound events to occur on different days. Figure 5.7 illustrates the distribution of the time differences between these bounds for all requests. If the request is for a single ride, the difference is counted for as 0 in this plot. It is evident that the majority of rides occur on the same day, but a significant portion has a one-day difference. This distribution gradually decreases as the time difference increases. Further examination shows that rides with their corresponding bound on a different day occur relatively often around the weekend days.

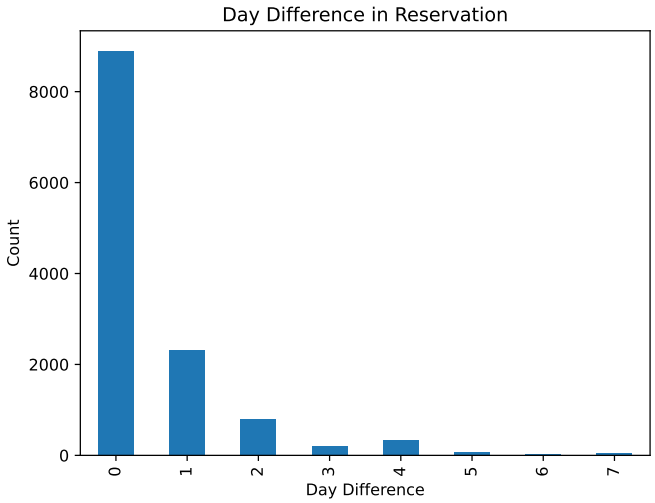

Figure 5.7: Difference in days outbound and inbound

Departure and Arrival Locations Furthermore, Figure 5.8 displays both the outbound and inbound request counts per departure location for pool locations and the ten most frequently occurring non-pool locations. The reason for the presence of non-pool locations within the departure locations is, as can be read in Sections 3.2.3 and 5.2.2, that in mandatory returns the outbound ride goes to a non-pool location, while the inbound ride arrives at a pool location. Therefore, the inbound ride departure locations include non-pool locations.

Plotting the request counts per arrival location, would give a very similar plot as the majority of the trips are returns, as shown in Figure 5.9. If all trips were returns, the plots of request counts per departure location and request counts per arrival location would be identical, as each departure location would always be an arrival location and vice versa. However, due to a number of single rides, there is a slight variation.

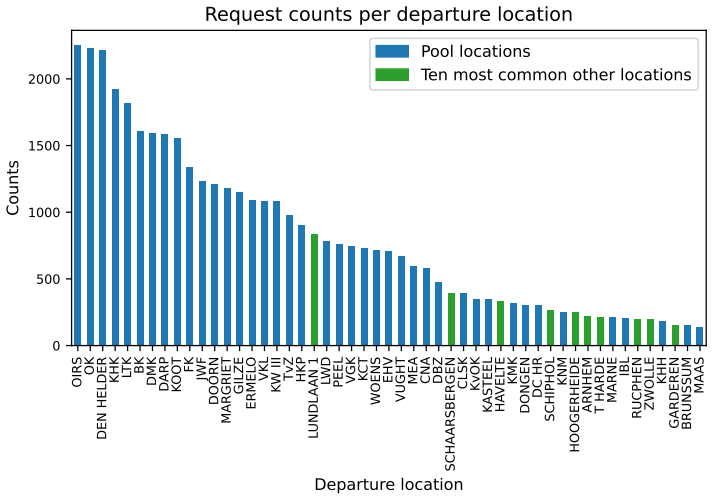

Figure 5.8: Request counts per departure location in Janaury '22

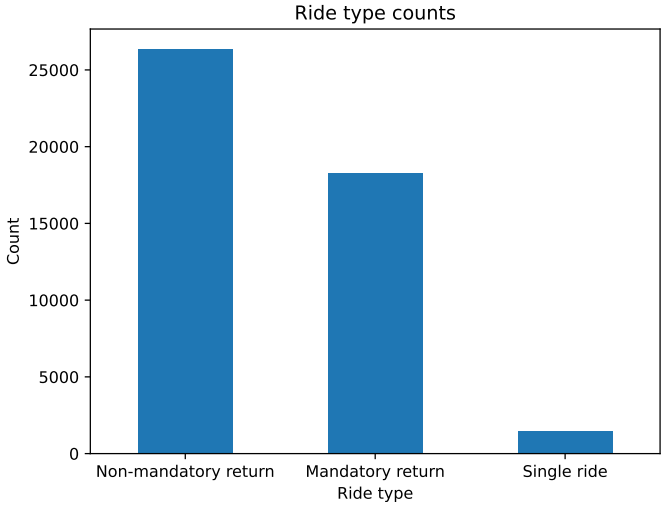

Figure 5.9: Ride type counts

An interesting observation is the relationship between the request counts per departure location and the number of available cars per pool location, the former in Figure 5.8 and the latter in Figure 5.2. By plotting these values against each other in a scatter plot with a trend line, shown in Figure 5.10, it is evident that a clear correlation exists. The number of vehicles per pool location is proportionate to the number of departure requests per pool location in January '22.

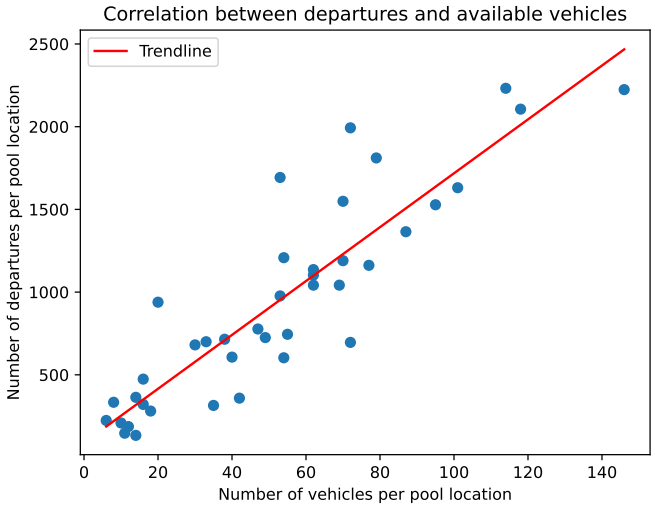

Figure 5.10: Correlation between departures and available vehicles

Time windows and Maximum Ride times Figure 5.11 presents the earliest departure and latest arrival times for outbound rides, categorized into hourly bins, encompassing all hours of a day for all requests in the month of January. It can be observed that outbound rides start as early as 4:00 AM, with a significant increase from 5:00 AM onwards. This trend continues to rise, and a clear peak is visible in the morning around 7:00 AM for both departures and arrivals of outbound rides. This pattern aligns with the rush hour when employees leave for appointments. At 8:00 AM, there is a peak in arrivals, indicating that most individuals have reached their appointments. Outbound rides decrease significantly after 10:00 AM, remaining relatively low throughout the day. Furthermore, it can be noted that, in general, there are more outbound departures than arrivals before 7:00 AM, which switches after 7:00 AM. Figure 5.12 displays a mirrored plot, representing the earliest departure and latest arrival times for inbound rides. In this figure, inbound rides increase throughout the day, reaching a peak around 4:00 PM and 5:00 PM for the earliest departure time, and a peak around 6:00 PM for the latest arrival time. Again, these findings align with the rush hour when people return home from their appointments. When combining these two figures in Figure 5.13 for both inbound and outbound rides, the distribution clearly illustrates the morning and evening rush hour peaks.

Figure 5.14 showcases a histogram representing maximum ride times for the requests placed in January, grouped into 5-minute bins. There are a few requests from the Netherlands to Norway, which resulted in a peak in the graph at a ride time of approximately 1200 minutes. These have been omitted from this plot for

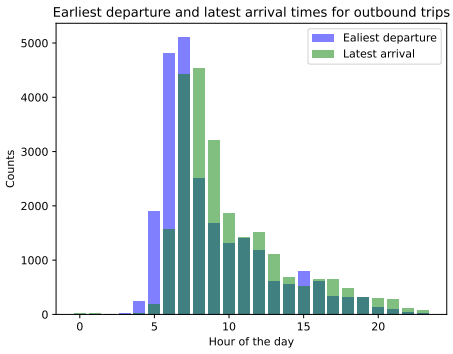

Figure 5.11: Earliest departure and latest arrival times for outbound rides

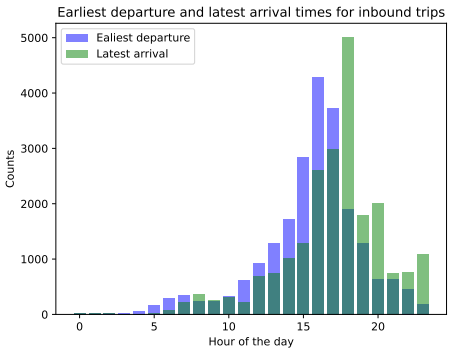

Figure 5.12: Earliest departure and latest arrival times for inbound rides

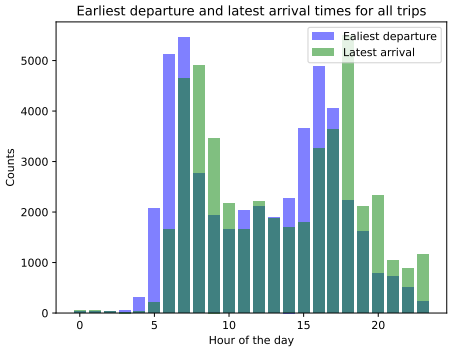

Figure 5.13: Earliest departure and latest arrival times for all trips

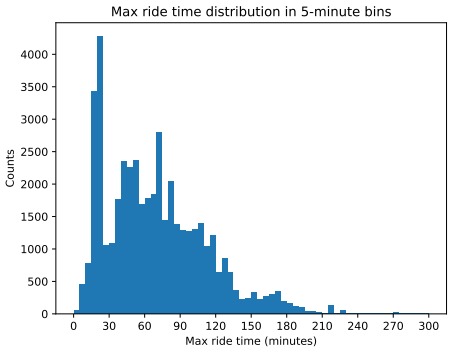

Figure 5.14: Max Ride Time Distribution in 5-minute bins

a clearer illustration. From this figure, it can be observed that there is a peak at approximately 20 minute-rides. After this peak, it increases steadily with a peak around 40 minute-rides and starts decreasing after. One more sudden peak happens at approximately 70-minute rides. The decrease continues steadily until approximately 200 minutes, indicating that rides might take more than 3 hours.

5.3 Distance Matrix

In the revised database, we are left with 676 unique locations. For algorithmic purposes, it is necessary to determine all pairwise distances and nominal travel times and store these in a distance matrix. This results in a total of $676^2 = 456,976$ distances and travel times. To handle such a large volume of data, a suitable software option is the *Google Distance Matrix API*, which provides up to 40,000 requests for free. However, this number is insufficient to fill our distance matrix completely. To address this issue, we utilize a portion of the available requests to directly populate the distance matrix, and for the remaining entries, we will employ estimation techniques as further described in this section.

5.3.1 Filling frequent entries

The entries being filled via API-requests, are the most frequently occurring locations, which include both pool locations and the most common non-pool locations. In this process, we can exploit the symmetry of the distance matrix. As the distances and travel times are symmetric along the diagonal and are zero on the diagonal, we do not need to make a request for each entry. For a total of 234 locations, comprising all 41 pool locations and the 193 most common locations, we only need $\frac{54,756-234}{2} = 27,261$ requests to fill the entries in the matrix, instead of $234^2 = 54,756$. After doing this, we are left with 12,739 requests.

5.3.2 Filling remaining entries

To complete the remaining entries in the distance matrix, we utilize the geographic coordinates of all locations and the remaining API-requests. A regression analysis is performed between the straight-line distances calculated from the coordinates and the actual distances obtained from API-requests. Besides, a second regression is performed between the actual distance and the actual travel time. From the Python library *Geopy*, the geocoding provider *Nominatim* is used to retrieve the coordinates of all locations. These coordinates enable the calculation of straight-line distances using the `Geodesic`-module also provided by the *Geopy*-library, which accounts for the Earth's curvature. We randomly select 5,000 pairs of locations and calculate their straight-line distances. Subsequently, we query the API for the corresponding distance and travel time. This process gives us the straight-line distance, and the exact travel distance and time provided by the API, for these 5,000 location pairs. Of these 5,000 data points, 80% are used for training a linear regression model, while the remaining 20% are used for testing. Note that a number of locations were not recognized by the API and are omitted from the regression.

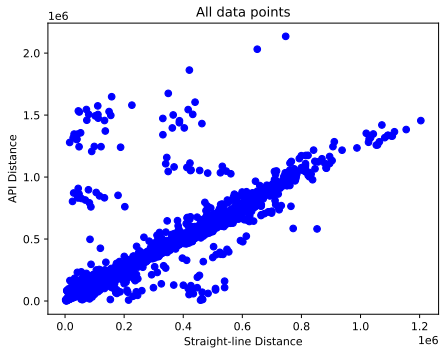

Figure 5.15: All regression data points of Straight-line Distance against API-Distance

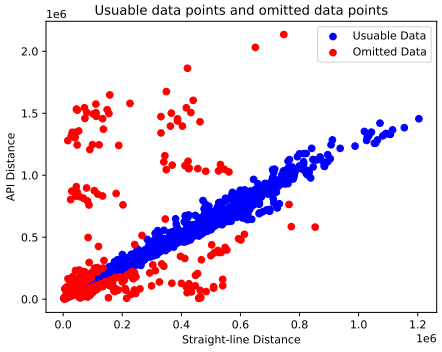

Figure 5.16: Usable and omitted regression data points of Straight-line Distance against API-Distance

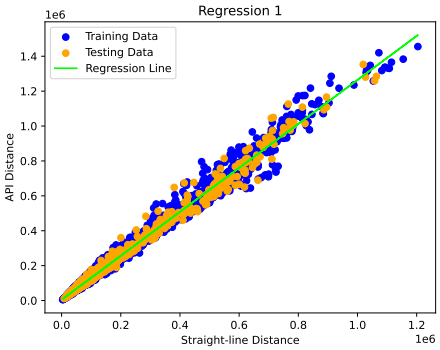

Figure 5.17: Training data, test data, and the regression line

Regression 1 The first regression is the regression between the straight-line distances calculated from the coordinates and the actual distances obtained from API-requests. In Figure 5.15, all data points are shown. In this figure, it is immediately apparent that there is a cloud of data points where a regression can be found. However, it is also noticeable that some points deviate from this cloud. The reason behind this discrepancy lies in the fact that location names are not always unique. For instance, there is a place called 'Bergen' in both the Netherlands and Germany. It is possible that when retrieving coordinates, 'Bergen' in the Netherlands is used, while the API considers 'Bergen' in Germany. To address this issue, API distances that are greater than 1.8 times the straight-line distance are omitted. Additionally, points where the API distances are smaller than the straight-line distance are also omitted. This results in a usable set of data points, which is depicted in Figure 5.16. Now, only the cloud remains, which data points are used in the regression. In this regression analysis, 80% of the data points are used as training data, while

Intercept	4440.834
Slope	1.260
R-squared	0.988

Table 5.1: Intercept, slope and R-squared of regression 1

the remaining 20% serve as test data. Figure 5.17 illustrates the training data, test data, and the regression line. Table 5.1 presents the obtained regression coefficients and the corresponding R-squared score. With all coordinates known, the straight-line distance can be calculated for all remaining pairs of the distance matrix, and an accurate estimation of the actual distance can be made using the following formula, with all distances in meters:

$$\text{Travel distance} = 4440.834 + 1.260 \times \text{straight-line distance}$$

This formula obtains an exceptionally high R-squared score of 0.988, indicating that approximately 98.8% of the variability in the API travel distance can be explained by the linear relationship with the straight-line distance.

Regression 2 In the second regression, we examine the actual driving distance and the actual travel time. In this case, we do not use the coordinates, and thus it is a regression between the two values directly provided by the *Google Distance Matrix API*. In this analysis, we once again focus on the 5,000 randomly selected location pairs that were queried with the API to obtain actual travel time and travel distance. As before, the pairs in which a location is not recognized by the API, are excluded. Figure 5.18 displays all the data points of distance against travel time. As before, we observe a distinct cluster of points where a regression can be identified. However, there are also data points that appear to deviate from the cluster, lying outside of it. These points correspond to pairs involving at least one of the following locations: 'Deil', 'Bergen', and 'Norway'. It is possible that the API utilizes different routing algorithms or data sources for calculating distances and travel times, which can account for these variations. To address this, we omit these points, resulting in the data points shown in Figure 5.19. With the filtered data points, we proceed to perform regression analysis on the remaining cluster. Once again, we allocate 80% of the data as training data and 20% as test data. Figure 5.20 shows the training data, test data, and regression line. In Table 5.2, the intercept, slope and R-squared are shown of this regression.

Intercept	702.494
Slope	0.0369
R-squared	0.998

Table 5.2: Intercept, slope and R-squared of regression 2

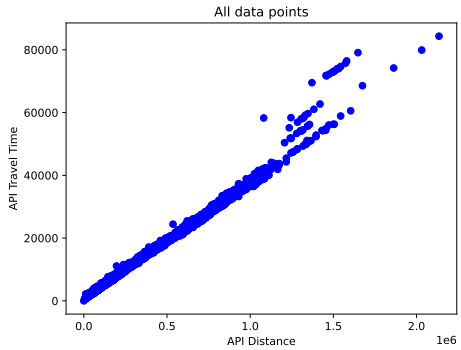

Figure 5.18: All regression data points of actual distance against actual travel time

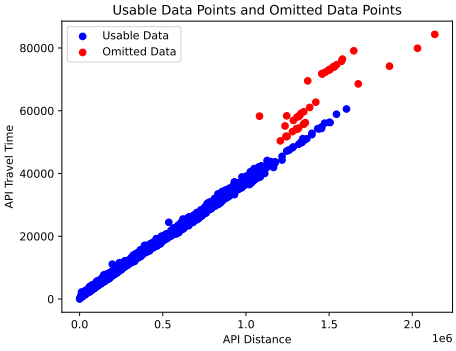

Figure 5.19: Usable and omitted regression data points of actual distance against actual travel time

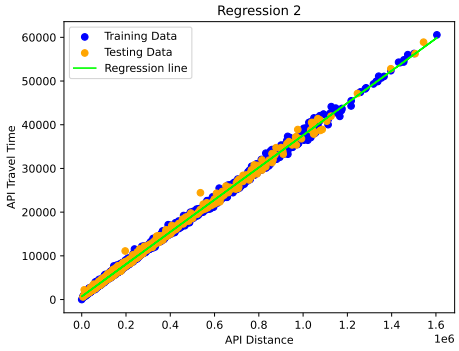

Figure 5.20: Training data, test data and the regression line

This results in the following formula for determining the actual travel time from the actual distance, with travel time in meters and travel distance in seconds:

$$\text{Travel time} = 702.494 + 0.0369 \times \text{travel distance}$$

Again, in this regression, an exceptionally high R-sqaured score is obtained. The score indicates that approximately 99.8% of the variability in the API travel time can be explained by the linear relationship with the API distance, providing a reliable formula for calculating the travel time from the obtained travel distance.

5.4 Recap on Data Preparation and Distance Matrix

In this section, we present a recap of the data preparation process and the formation of the distance matrix, along with its intended application.

Data Preparation In Section 5.2.2, we discussed the processing of request data to make it usable for developing and evaluating the planning algorithm in Section 8, in the context of the new situation described in Section 3.2. Historic request data of January '22 was employed for this purpose. The initial step involved the removal of unusable rows and columns. Subsequently, string similarity algorithms were utilized to group similar manually entered locations. Furthermore, the different ride types were introduced, and returns were split into two separate requests in the database. Finally, we explained the procedure for establishing time windows for these requests.

Distance Matrix In Section 5.3, we discussed the process of forming the distance matrix. We employed the *Google Distance Matrix API* to retrieve distances and ride times. Due to the limited number of requests allowed by the API, the distance matrix entries corresponding to important locations were directly obtained through requests to the API, and estimations were made for the remaining entries. To accomplish these estimations, two regression models were developed: **Regression 1** for travel distance and **Regression 2** for travel time. For each regression, a dataset comprising 5,000 random location pairs was used, with 80% serving as training data and 20% as test data. The API was employed to obtain the travel time and distance for all these location pairs. Additionally, the coordinates of all locations present in the distance matrix were obtained.

Regression 1 yields a formula to estimate travel distance based on the straight-line distance derived from the coordinates. On the other hand, Regression 2 provides a formula to estimate travel time based on travel distance. Both regression models demonstrate an exceptionally high R-squared score. Consequently, the missing entries in the distance matrix are reliably estimated using the formula from Regression 1 to estimate travel distance based on coordinates, and then employing the formula from Regression 2 to estimate travel time using the obtained travel distance.

Chapter 6

Solution Method

In this section, we present the solution method for addressing the ride-sharing problem for the situation discussed in Section 3.2. The algorithm employed to solve this problem is referred to as `CombinedApproach`, named so because it considers both ride-sharing possibilities and the option of vehicles operating independently of their pool location. In Section 6.1, we present the assumptions and simplifications the model makes. Moreover, Section 6.2 illustrates how the problem is represented and implemented. The preprocessing steps involved in this algorithm will be examined in Section 6.3. Solving the problem encompasses two distinct aspects: the initial solution heuristic, described in Section 6.4, and an iterated local search (ILS) which uses a simulated annealing (SA) approach, described in Section 6.5. Additionally, we will elaborate on the neighbourhood operators and demonstrate how the constraints, discussed in Section 4.2, are checked.

6.1 Assumptions and Simplifications

Due to the complexity of the real-world situation, `CombinedApproach` relies on certain assumptions and simplifications to facilitate the analysis. These are outlined below:

- There are no uncertainties and disturbances. However, we revise this assumption in Sections 8.3 and 9.3.
- The nominal travel time and distance between locations are realistic, but constant, pre-known, and do not change. In Section 5.3, we described how we got this data.
- The need for vehicle charging is not considered.
- Time windows and maximum ride times are artificially established due to absence of this data, as discussed in Section 5.2.2.

6.2 Representations and Implementations

In Section 6.2.1, we will discuss the representation and implementation of the problem. In Section 6.2.2, we specify how routes are presented.

6.2.1 Graph

The problem is represented as a weighted directed graph. We will discuss the nodes and arcs and what they present.

Nodes A request consists of two events: a pickup and a dropoff. Both are represented as nodes in the graph. It should be noted that in Section 5.2.2, it was described that return requests are split into two separate requests with reversed locations. Therefore, a user's return would be presented as four nodes in the graph: a pickup and dropoff for the outbound ride, and a pickup and dropoff for the inbound ride. Nodes have properties, i.e. attributes, which are listed and shortly explained below:

- **Node ID** is a unique combination of strings and digits to represent the request ID, whether it is a pickup or dropoff node, whether it is a node that corresponds to an inbound or outbound ride, and whether it is a node that corresponds to a single, mandatory or non-mandatory return.
- **Location** represents the specific location of the node.
- **Type** represents whether the node is a pickup or a dropoff.
- **Neighbours** represents a list of neighbours its arcs are directed to. Shortly, we will elaborate on the arcs and explain why they are important.
- **Time window** represents the time window of the corresponding activity of the node. This could be a departure time window for pickups and an arrival time window for dropoffs. Construction of time windows is explained in Section 5.2.2.
- **Event time** is a moment in time that corresponds to the actual time the corresponding event would occur. The event time of a node is determined based on the event time of the previous node and the travel time from the previous node to the current node, as explained in Section 4.2.3.
- **Maximum ride time** presents the maximum ride time of the corresponding user. This concept is explained in Section 5.2.2.

Arcs An arc is established from node A to node B if it is potentially feasible, within the given time windows, for a vehicle to handle event A first, and event B subsequently. All nodes are initially connected to each other, forming a complete graph. However, to adequately form the graph, the arc from a starting node to an ending node is pruned in the following cases, as described by Cordeau [18] (2006):

- The starting node represents a dropoff, and the ending node represents the corresponding pickup.
- The starting node represents a node from the inbound ride, and the ending node represents a node from the corresponding outbound ride.
- The earliest departure time of the starting node, plus the travel time to the ending node and the service time, exceeds the latest departure time of the ending node.

Note that we will elaborate on an additional pruning rule, which is based on detours and maximum ride times, in Section 6.3

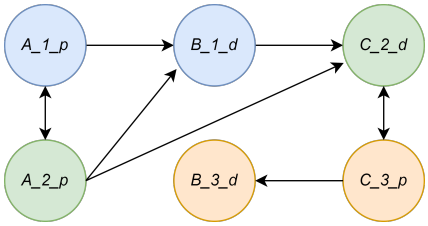

Figure 6.1: Example of a graph for pickups and dropoffs of single rides of person 1, 2, and 3, and locations A, B, and C

In Figure 6.1, we present an example of a graph illustrating the pickup and dropoff nodes associated with three single-ride requests from individuals 1, 2, and 3. The nodes in this graph represent three locations: A, B, and C. Notably, each pickup node is connected to its corresponding dropoff node, indicating the possibility of a vehicle handling the dropoff event right after the pickup. Furthermore, the figure demonstrates the presence of arcs between events that are not linked to the same individual, for instance, the two events occurring at location A: the pickup of person 1 and the pickup of person 2. This implies the potential feasibility of executing both events in sequence using a single vehicle. An example of a potentially feasible route in this graph is:

$$A\_1\_p \rightarrow A\_2\_p \rightarrow B\_1\_d \rightarrow C\_2\_d \rightarrow C\_3\_p \rightarrow B\_3\_d$$

Initially, the vehicle is located at pool location A, and person 1 takes the vehicle for their single trip. At location A, person 2 also boards the vehicle. They then

travel together to location B, where person 1 is dropped off. Person 2 continues alone to location C, where their trip ends. At location C, the vehicle is taken by person 3, who drives their trip to pool location B. Thus, a total of three trips are completed. Notice how ride-sharing occurs between individuals 1 and 2 in this route, and how the vehicle is not bound to a specific pool location, as the route starts at location A and ends at B.

Furthermore, it is important to note that every feasible route is present as a path in the graph, but that not every route found in the graph is necessarily feasible, as we will explain shortly. This is why we previously used the term 'potentially feasible'. As this is an important observation, we will state it here:

Graph Paths: A route can only be considered feasible if it exists as a path in the graph. However, it is important to note that not every path present in the graph is a feasible route. For a route to be feasible, it must also adhere to all the constraints outlined in Section 4.2.

We explain this concept in Figure 6.2. We see that the earliest departure time from the starting node i, added to the travel time to the ending node j and the service time, does not exceed the latest departure time of node j. Therefore, in the corresponding graph, an arc would be present from node i to node j. However, the event time, discussed earlier in Section 4.2.3, may not align precisely with e_i, but rather with $e_i + \delta$, as shown in the figure. In this scenario, the departure time plus the travel time to the ending node and the service time, exceeds the latest departure time of the ending node. Consequently, it is not possible to handle event j after event i with an event time like this at node i, despite the presence of an arc in the graph connecting the two nodes.

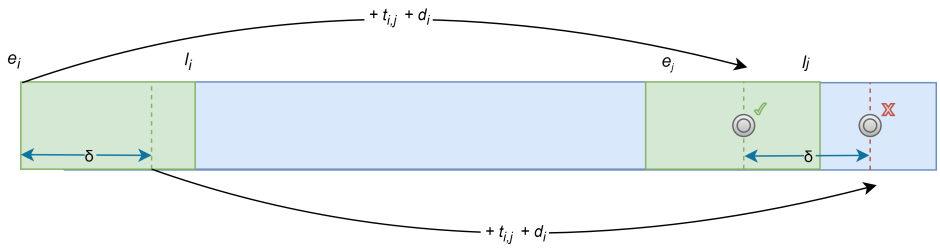

Figure 6.2: Unfeasible event handling

Moreover, the weight of an arc is a tuple representing the travel distance and travel time from the location that corresponds with the pickup to the location that corresponds with the dropoff. These values can be derived from the distance matrix, which is described in Section 5.3. Note that these weights are not present in Figure 6.1 for oversight.

6.2.2 Routes

As described before, in Section 4, a route denotes the ride plan for a single vehicle. A route is represented as a list of nodes. Furthermore, a route has the following properties, i.e. attributes:

- **Route ID** is a unique number to identify each route, and thus vehicle.
- **Route type** represents the type of a route. As described before, in Section 4, a route can have either type *original* or *additional*. The original routes refer to the routes that constitute the final solution, representing the actual routes to be driven. Additional routes, on the other hand, are scheduled for dummy vehicles. Similar to the original routes, these routes adhere to all constraints, except Constraint **1.**, which states that a route should start and end at a pool location. These additional routes have a maximum capacity of eight.

6.3 Preprocessing

In the initial complete graph, arcs between nodes are pruned based on the criteria described before in Section 6.2.1. Additionally, further pruning can be conducted following another rule proposed by Cordeau [18] (2006). Specifically, consider a pickup node, denoted as i and its corresponding dropoff node, denoted as $n + i$. Arcs (i, j) and $(j, n + i)$ with any other node j are both pruned if the travel time from i to j, plus the service time at j and travel time from j to $n + i$, exceeds the maximum ride time specified by the person associated with node i. In Section 5.2.2 is explained how the maximum ride time is determined.

In Figure 6.3 and Figure 6.4, histograms are presented that show respectively the distribution of the number of neighbours per node before and after applying this pruning rule, for the typical day of January 18th. The number of neighbours is categorized into bins of size 100. Before pruning, the histogram seemingly exhibits two peaks. The first peak occurs around 1500 neighbours, followed by a decrease and then a subsequent peak around 5500 neighbours. The distribution does not display a clear pattern but rather a mixture of various values. However, after pruning, a significant reduction in the number of neighbours per node is observed. The distribution is no longer scattered, but rather a peak is formed in the range of 0 to 100 neighbours, gradually decreasing until reaching a single node with 4000 neighbours. This contrast with the situation before pruning, where certain nodes had as many as 7000 neighbours. Nodes with such a high number of neighbours typically correspond to those with an extremely early earliest departure time, ensuring that the earliest departure time of the starting node, plus the travel time to the ending

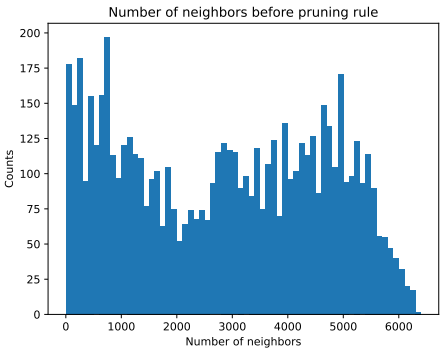

Figure 6.3: Number of neighbours per node before the pruning rule

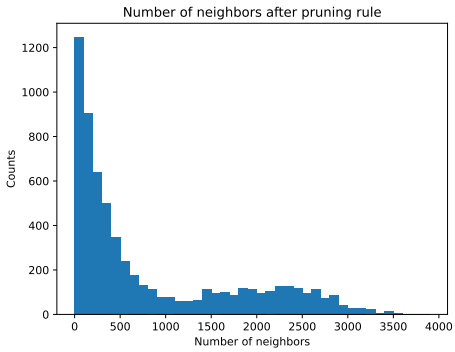

Figure 6.4: Number of neighbours per node after the pruning rule

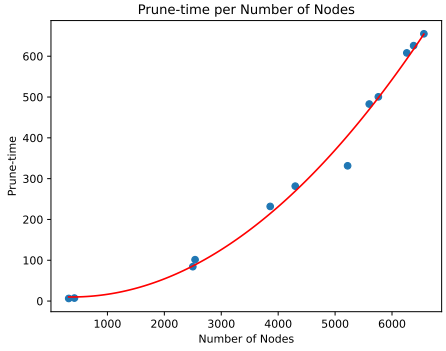

Figure 6.5: Prune time for different number of nodes

node and the service time, does not exceed the latest departure time of the ending node, as discussed in Section 6.2.1.

The time complexity of this graph pruning rule is quadratic in the number of nodes, as for each node, every other node is examined. This relationship is shown in Figure 6.5, where the time required for the pruning is plotted against the number of nodes in the graph. A quadratic polynomial curve has been fitted to the data points, highlighting the quadratic behaviour of this pruning process.

6.4 Initial Solution Heuristic

After the preprocessing stage, the matching stage begins, wherein the first step involves searching for an initial solution. The insertion heuristic used for this purpose is described in this section. Subsequently, Section 6.5 elaborates on the subsequent local search, focusing on the exploration of neighbourhood operators and constraint checks. Before delving further into these concepts, it is important to highlight the following statement:

Only Feasible Routes: The solution exclusively contains feasible routes, i.e., routes that adhere to all the constraints outlined in Section 4.2.

The initial solution heuristic for `CombinedApproach` is based on the construction insertion heuristics provided by Braekers, Caris, and Janssens [12] (2014), and Masmoudi, Hosny, Braekers, and Dammak [40] (2016). The pseudocode presented in Algorithm 1 describes this heuristic. In this algorithm, all positions in all routes are considered, and nodes corresponding to requests are inserted at their best feasible positions. Their best position is determined using the objective function described in Section 4.1, and a feasible route adheres to all constraints described in Section 4.2. Note that returns can be divided into an outbound and inbound ride, and that in the case of a return request therefore four nodes need to be placed. For single ride requests, two nodes need to be placed. In the following paragraphs, we will discuss the different components of Algorithm 1.

Algorithm 1 Initial Solution Heuristic for `CombinedApproach`

```
1:  Initialize empty routes
2:  for mandatory in requests do
3:      PossiblePlacements = GetPossiblePlacements(outbound, inbound)
4:      if not PlacedInBestPosition(PossiblePlacements) then
5:          AddAdditional(outbound, inbound)
6:      end if
7:  end for
8:
9:  for non-mandatory in requests do
10:     PossiblePlacements = GetPossiblePlacements(outbound)
11:     if not PlacedInBestPosition(PossiblePlacements) then
12:         AddAdditional(outbound, inbound)
13:     else
14:         PossiblePlacements = GetPossiblePlacements(inbound)
15:         if not PlacedInBestPosition(PossiblePlacements) then
16:             AddAdditional(inbound)
17:         end if
18:     end if
19: end for
20:
21: for single rides in requests do
22:     PossiblePlacements = GetPossiblePlacements(outbound)
23:     if not PlacedInBestPosition(PossiblePlacements) then
24:         AddAdditional(outbound)
25:     end if
26: end for
```

6.4.1 Initialization

As can be seen in Algorithm 1, first the initialization of empty routes for available vehicles per pool location takes place. After doing this, the solution exists of empty original routes only, which all correspond to a real vehicle. After that, the requests are divided into three categories, described in Sections 3.2.3, 4.2.1, and 5.2.2, corresponding to their ride types: mandatory return requests, non-mandatory return requests and single ride requests. All requests in these three categories are placed in routes in the respective order.

6.4.2 Order of Placement

This placement order is based on the difficulty of combining the request type in a ride-sharing arrangement. Mandatory return requests are the greatest challenge due to their limited variability and are thus handled first. Firstly, the inbound ride must always occur when the outbound ride takes place. Secondly, the likelihood of ride-sharing is lower for these requests since they involve non-pool locations often scattered across the country. After the mandatory return requests, non-mandatory return requests and single rides are addressed, which allow for more variability. In non-mandatory cases, the return can be split into two single rides, as we will discuss soon, making it easier to place them in a route.

6.4.3 Function Explanations

Algorithm 1 makes use of various functions for different procedures. This paragraph introduces and discusses these functions.

Get possible placements Algorithm 1 frequently utilizes the function `GetPossiblePlacements()`, which can be called with only an outbound, only an inbound, or both. This function is shown in Algorithm 2. Depending on the input it receives, the function behaves differently.

In the case of only an outbound or inbound, i.e. when two nodes need to be placed, the function operates as follows: all possible placement positions are considered for the pickup and dropoff within each route. Next, it is checked in `PathInGraph()` if the resulting placements lead to a route which corresponds to a valid path in the graph, as a route is only potentially feasible if it occurs as a path in the graph, as described in Section 6.2.1. More on this function is described in the next paragraph. Furthermore, it is crucial to note that the actual placement of nodes does not occur during this verification process. If the path is present in the graph, the placement option is added to a list named PossiblePlacements.

In the case where both outbound and inbound nodes are provided, the function examines whether all four nodes can be placed together either before or after a route. This is checked separately in the functions `PlacementBefore()` and `PlacementAfter()`, which take the two pickups and two dropoffs as inputs. The possible placements resulting from these checks are also added to the list.

Finally, the function returns the list with possible placements. These potential placement options may be feasible as the paths occur in the graph, but it is not checked yet whether they adhere to the constraints discussed in Section 4.2. The constraint checks will be discussed in detail in Sections 6.5.2 and 6.5.3. Furthermore, note that the function also takes into account empty routes during this evaluation process.

Algorithm 2 Function to Get Possible Placements

```
1:  function GetPossiblePlacements(outbound, inbound)
2:      possiblePlacements = []
3:      if outbound xor inbound then
4:          for route in routes do
5:              for i in range(0, length(route) - 1) do
6:                  for j in range(i, length(route)) do
7:                      if PathInGraph(i, j, pickup, dropoff, route) then
8:                          possiblePlacements.append((route, i, j))
9:                      end if
10:                 end for
11:             end for
12:         end for
13:     else if outbound and inbound then
14:         for route in routes do
15:             if PlacementBefore(pickups, dropoffs, route) then
16:                 PossiblePlacements.append((route, 0))
17:             end if
18:             if PlacementAfter(pickups, dropoffs, route) then
19:                 PossiblePlacements.append((route, length(route)))
20:             end if
21:         end for
22:     end if
23:     return PossiblePlacements
24: end function
```

Path in Graph A route is only potentially feasible if it occurs as a path in the graph, as described in Section 6.2.1. Therefore, we will only consider placements for which the resulting route occurs as a path in the graph. We know that all routes in the solution are present in the graph, as only feasible routes are accepted. Therefore, it is necessary to validate only the part of the path that is affected when

a node is added to the route. These affected nodes are the ones directly linked to the placement position, i.e., the preceding and succeeding nodes. Consequently, to check the placement of a pickup at position i and a dropoff at position j, it is essential to verify whether the node at position i is a neighbour of the node at position $i-1$, and the node at position $i+1$ is a neighbour of the node at position i. Similarly, it must be confirmed whether the node at position j is a neighbour of the node at position $j-1$, and the node at position $j+1$ is a neighbour of the node at position j. Here, the node attribute representing neighbours, discussed in Section 6.2.1, comes into play.

Placed in Best Position The returned list by `GetPossiblePlacements()`, is used as input for `PlacedInBestPosition()`, as shown in Algorithm 1. This function is shown in Algorithm 3. Here, feasibility checks and actual placement of nodes are performed. More specifically, the best placement option is popped from the list of possible placements and inserted, as long as the best is not feasible. The function returns a boolean indicating whether a feasible placement has occurred or not.

Algorithm 3 Function to place nodes in best position in existing route

```
1:  function PlacedInBestPosition([PossiblePlacements])
2:      Placement = False
3:      while [PossiblePlacements] do
4:          [PossiblePlacements].pop(Best)
5:          if Best is feasible then
6:              Execute best placement
7:              Placement = True
8:              break
9:          end if
10:     end while
11:     return Placement
12: end function
```

Add Additional Vehicle Another function frequently used in Algorithm 1 is `AddAdditional()`, which can also be called with only an inbound, only an outbound, or both. This function creates a new additional route in which the nodes corresponding to the provided bounds are placed.

6.4.4 Placement

This paragraph discusses the placement of the three types of requests. We will address them in the order in which they are placed.

Placement of Mandatory Returns As explained and shown in Algorithm 1, after categorization, first the mandatory return requests are considered for placement. In handling these, `GetPossiblePlacements()` is called with both outbound and inbound. As previously described, the four corresponding nodes are always placed consecutively in a route, following the order of pickup outbound, dropoff outbound, pickup inbound, and dropoff inbound. If no placement is possible, an additional route is created to accommodate the four nodes. Placing the four nodes of a mandatory return together in one route, facilitates their relocation during a local search process. Separating the outbound and inbound legs and placing them both in additional routes makes it highly unlikely for them to be reassigned to original routes because of the used operators, as we will shortly see in Section 6.5.1.

Placement of Non-mandatory Returns Next, nodes of non-mandatory requests are considered. We split the outbound and inbound and look at both independently. Initially, only the outbound is considered when calling GetPossiblePlacements(). If no feasible placements are found, both the outbound and corresponding inbound nodes are placed in a new additional route. If the outbound is placed in an original route, both additional and original routes are considered for the inbound. However, when the best position for an outbound is an additional route, the inbound can only be placed in an additional route as well. If no feasible location is found for the inbound, it is added to a new 'additional' route. This approach let us satisfy Constraint **4.**.

Placement of Single Rides Single rides are placed similarly as the outbound ride of non-mandatory returns. Placements are considered, and if no feasible placement is found, the nodes are placed in a new additional route. Note that, the category of single rides also includes outbound and inbound rides where the corresponding inbound or outbound ride is planned on a different day. These rides can be processed as single rides as long as they adhere to Constraint **4.**. The reason for this is only two nodes are handled instead of four.

6.4.5 Constraint Checks

All routes for which constraints need to be examined are potentially feasible, as we are sure the path is present in the graph. However, to ensure feasibility, the route must also adhere to the constraints outlined in Section 4.2. In this paragraph, we explain how we ensure routes adhere to these constraints.

- **Constraint 1.** If four nodes are placed in a route together, the first and last node of four always correspond to pool locations. This remains true even in

the case of a mandatory return, where the second and third node are non-pool locations. In case of a single ride and non-mandatory return, all nodes correspond to pool locations, so it does not matter where the nodes are placed in the route. As a result, Constraint **1.** is always satisfied in this heuristic.

- **Constraint 2.** As all pickups and corresponding dropoffs are always placed together, Constraint **2.** is satisfied.
- **Constraints 3. and 5.** It can either be the case that two nodes or four nodes are placed, depending on the request corresponding to a single ride, a mandatory, or a non-mandatory return. When two nodes are placed, the pickup node is always placed first, followed by the dropoff node in the route. When four nodes are placed in the same route, the relative order is always as follows: outbound pickup, outbound dropoff, inbound pickup, and inbound dropoff. This satisfies both Constraints **3.** and **5.**. Note that there is a possibility of outbound and inbound nodes being placed in different routes. By adhering to the designated time windows, Constraint **5.** is also satisfied in such cases.
- **Constraint 4.** As described before, in Algorithm 1 first the outbound nodes are placed when handling non-mandatory returns. Depending on the type of route these are placed in, the type of the route the inbound nodes can be placed in is determined. Mandatory returns involve placing all nodes within the same route. Because of these approaches in both mandatory and non-mandatory returns, Constraint **4.** is satisfied.
- **Constraints 6., 7., 8. and 9.** The more complex constraint checks, i.e. Constraints **6.**, **7.**, **8.**, and **9.**, which require updates of data structures, are discussed in Section 6.5.3.

6.5 Local Search

The local search method used for improving the initial solution is simulated annealing (SA). In the experiments in Section 8, we will use SA in an iterated local search (ILS). More on ILS is discussed in Section 7.6. The SA pseudocode is presented in Algorithm 4. Note that both original and additional routes are considered in the search process. The first step involves initializing the hyperparameters: starting temperature T_0, stopping temperature T_{stop}, current temperature T, the number of iterations per temperature N, the cooling scheduler c, the current solution S, and the best solution S_{best}. Temperature T controls the probability of accepting

a worse solution during the optimization process, and it gradually decreases during the annealing process by multiplying it with c. The number of iterations per temperature N determines how many iterations are performed at each temperature stage before decreasing T further. In each iteration, the solution space is explored through the use of neighbourhood operators. Each solution is evaluated using the objective function, presented in Section 4.1. Better solutions are always accepted, while worse solutions are accepted with a decreasing probability, thereby reducing the likelihood of accepting worse solutions. Four distinct neighbourhood operators are utilized, denoted as I1, I2, I3, and I4, which will be explained in Section 6.5.1. Additionally, the process of verifying feasibility constraints will be discussed in Sections 6.5.2 and 6.5.3.

Algorithm 4 Simulated Annealing `CombinedApproach`

Initialize T_0, T_{stop}, c
Initialize temperature T=T_0
Initialize number of iterations per temperature N
Initialize current solution S using Initial Solution Heuristic
Initialize best solution $S_{\text{best}} = S$
$i = 0$
while $T > T_{\text{stop}}$ **do**
    while $i < N$ **do**
        I = random.choice({I1, I2, I3, I4})
        S_{new} = ApplyOperator(I, S)
        if Feasible(S_{new}) **then**
            δ = objective(S_{new}) - objective(S)
            if $\delta < 0$ or random.uniform$(0, 1) < e^{\frac{-\delta}{T}}$ **then**
                $S = S_{\text{new}}$
                if S is better than S_{best} **then**
                $S_{\text{best}} = S$
                end if
            end if
        elseUndoChanges(S_{new})
        end if
        $i++$
    end while
    $T = T \times c$
end while
return S_{best}

6.5.1 Neighbourhood Operators

In this section, we will discuss the neighbourhood operators. The constraint checks that verify the feasibility of a resulting route after applying an operator will be covered in Sections 6.5.2 and 6.5.3. However, it is worth noting that a check that occurs during the execution of the operators is the validation of the path in the graph, as described in Sections 6.2.1 and 6.4.3. Only if this check is successful, indicating that a route is potentially feasible, the other constraint checks will be performed. Although we will not explicitly mention this path check in the descriptions of the search operators, it does take place as a crucial preliminary step.

Swap I1 The first and simplest operator is the swap-operator, denoted as I1. The Swap-operator performs a straightforward operation that swaps the positions of two nodes within a single route. These nodes can be either pickup or dropoff nodes. Figure 6.6 provides a visual example of the swap operation. Specifically, node v_{n+i} and node v_j are swapped, resulting in the dropoff of i occurring after the pickup of j, rather than the original order. It is essential to note that a dropoff node may never precede its corresponding pickup node. Hence, it would for example not be possible for v_i and v_{n+j} to undergo a swap, as that would for both rides violate the constraint where the dropoff occurs before the pickup. Furthermore, although it is the case in Figure 6.6, it is important to note that it is not strictly required for the two nodes to be adjacent to be swapped. The swap operation allows for nodes to be interchanged regardless of their positions in the route, as long as all constraints are satisfied.

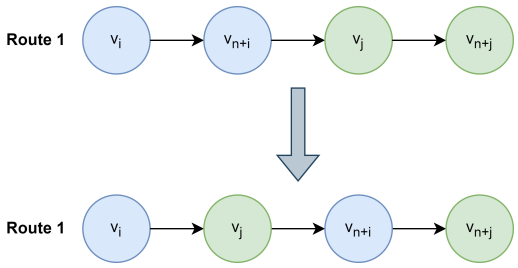

Figure 6.6: Swap-operator

Route exchange I2 The next operator is the Route Exchange-operator, referred to as I2. This operator selects a pickup node and its corresponding dropoff node from one route and relocates them to another route at the optimal position. The optimal solution is determined through evaluation with the objective function. The pickup is always placed before the dropoff, and all possible positions are checked and evaluated. Figure 6.7 provides an illustrative example. In this example, pickup

node v_j and its corresponding dropoff node v_{n+j} are removed from Route 1 and placed in Route 2 at their optimal positions. As a result, Route 1 is left with nodes v_i and v_{n+i}.

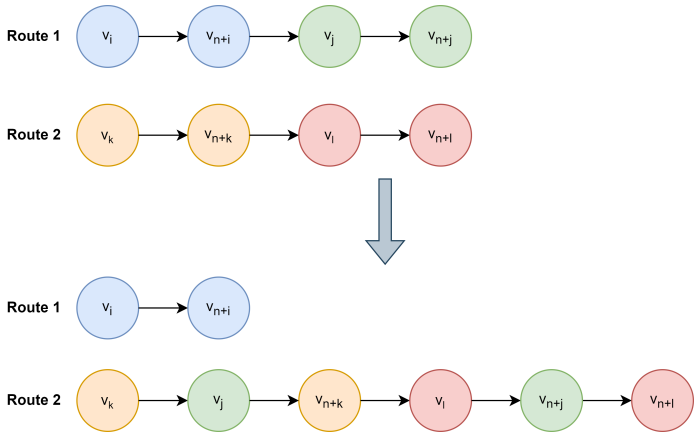

Figure 6.7: Route exchange-operator

Subroute Transfer I3 The Subroute Transfer-operator, denoted as I3, is a more complex operation compared to the Route Exchange-operator. Instead of removing a single pickup and dropoff from one route and placing them in another, the Subroute Transfer-operator involves transferring an entire subroute, which may consist of multiple pickups and dropoffs. It is important to note that the subroute must be independent, for the operation to be performed. As independency is an important definition, we will state it here.

Independent subroute: An independent subroute is a one-to-one mapping between pickups and their corresponding dropoffs within a subroute, ensuring that each pickup has its respective dropoff and there are no dropoffs without a corresponding pickup. Independent subroutes are always feasible routes on their own.

The property of independency plays an important role in feasibility checks, as we will see in Section 6.5.2. When this operator places a subroute in another route, the order of the nodes remains unchanged. The nodes within the subroute are not separated but rather kept intact, preserving their original sequence. Figure 6.8 provides a visualization of this process. In this example, the subroute consisting of nodes v_l, v_j, v_{n+l}, and v_{n+j} is removed from Route 1 and placed as a whole in Route 2, following the dropoff node v_{n+k}. It is important to note that this operation is only possible in this specific example if v_{n+k} and v_l occur at the same location, as there is no driver present in the vehicle to drive the vehicle from the location of v_{n+k} to v_l.

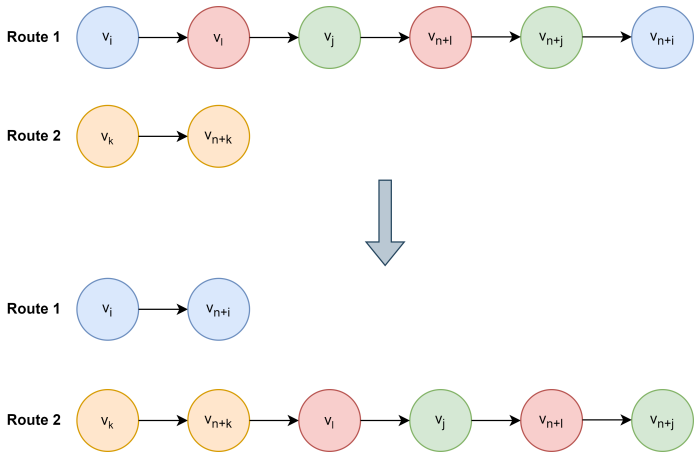

Figure 6.8: Subroute Transfer-operator

Tail-swap I4 The last operator is the Tail Swap-operator, referred to as I4. In this operator, the tails of two routes are exchanged. Both tails must be independent subroutes. This observation has important implications, as enlightened in Section 6.5.2. One of these implications is that it is guaranteed that at the breaking point of a head and an independent tail, no users are present in the vehicle, and the two points around the breaking point correspond to the same location. An extension to this observation is that if a tail swap can occur between two routes, all four nodes, including the last node before the tail and the first node of the tail for both routes, correspond to the same location. Figure 6.9 illustrates an example of the Tail Swap-operator in which both tails are a single pickup en dropoff pair. However, note that the tail could consist of multiple dropoffs and pickups. The independent subroute consisting of nodes v_j and v_{n+j} is swapped with the independent tail consisting of nodes v_l and v_{n+l} from Route 2. In this case, nodes v_{n+i}, v_l, v_{n+k}, and v_j all correspond to the same location.

Additional Routes Considering additional routes in the local search is crucial because it ensures that all requests are considered, even if they were not initially assigned to an original route. In the additional routes, feasible combinations in ride-sharing can also be explored using this approach, potentially placing them as a whole with operators I3 or I4 into an original route at a later stage. These additional routes also act as 'reserve bench', where requests can be placed during the search to help escape local optima and improve the optimization process.

Furthermore, in Section 6.4, we showed that when creating the initial solution, nodes from mandatory returns are always grouped together in a single route rather than being distributed across separate routes. Operators I3 and I4 clearly demonstrate why it is essential to have inbound and outbound nodes together within an

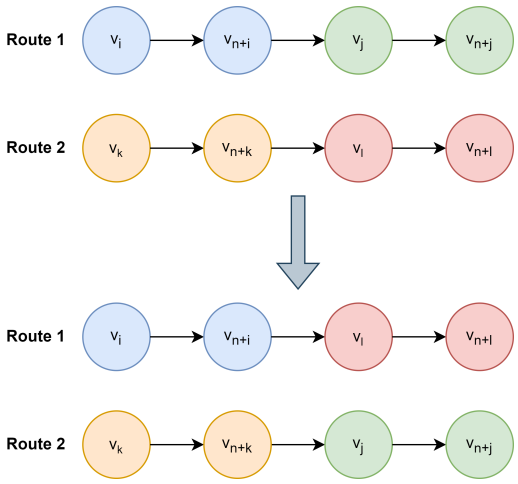

Figure 6.9: Tail Swap-operator

additional route at the initial solution: the grouping allows these operators to move these nodes as a whole. This can be useful when relocating the nodes to a route with another route type. If the outbound and inbound of a mandatory return were in separate additional routes, it would not be possible to move them back to an original route without violating Constraint **4.**. This constraint requires both the outbound and inbound nodes of mandatory returns to be in either an original route or an additional route. It would not be possible to move the outbound nodes to an original route first and then the inbound nodes separately, while adhering to this constraint. Both the outbound and inbound nodes must therefore be moved simultaneously.

6.5.2 Simple Constraint Checks

This subsection focuses on feasibility checks to ensure the routes remain feasible after the operators modify either one route, i.e. I1, or two routes, i.e. I2, I3, and I4. As described before, the feasibility of routes is assessed by verifying all the constraints mentioned in Section 4.2. Infeasible routes are not accepted as part of the solution. If a constraint is violated, and the route is thus found infeasible, the operator is terminated, the changes are undone, and the next iteration is started, as shown in Algorithm 4. Rather than executing all the checks at once after performing all the route updates, an approach that improves efficiency is adopted. Initially, simple constraint checks are performed. Updates that are needed for more complex constraint checks are executed only when all the preceding constraints are satisfied, and subsequently, the feasibility with these updates is assessed.

In this section, the simple checks that do not require updates are discussed. Those include most checks for the routing constraints, described in Section 4.2.1.

In Table 6.1, a scheme is presented to give insight in which constraints need to be checked for which operator. A green plane means the constraint does not need to be checked, while a red plane means it needs to be checked. It should be noted that in operators I2 and I3, two processes occur sequentially. First, nodes are extracted from one route, and then they are inserted into another route. Constraints need to be checked for both processes. Initially, all the checks related to the route from which the nodes are extracted are conducted, followed by checks for the route where the nodes are inserted.

- **Constraint 1.** When an operator modifies the first or last node of a route, it is necessary to verify if it corresponds to a pool location. If it does not, it is found infeasible. This ensures Constraint **1.** is satisfied. Note in Table 6.1 that this constraint needs to be checked for all operators, except for I4. The reason for this is that in the tail swap, no start point will change, so this does not to be checked. Besides, both tails are already in the solution and therefore both end locations are known to be feasible.

- **Constraint 2.** Constraint **2.** cannot be violated by any of the operators, and therefore does not need to be checked in any operator. For operator I1, this constraint is always satisfied, since nodes are not removed from a route. In operator I2, corresponding pickups and dropoffs are always grouped together, ensuring they remain together. As feasible subroutes are independent in I3 and I4, it is guaranteed the pickup and dropoff always occur together when replacing subroutes.

- **Constraint 3.** It is straightforward to enforce Constraint **3.**. The check in I1 is discussed in Section 6.5.3, as it requires an update. In operator I2, the dropoff is always placed after the pickup during the insertion phase, so no check is performed. In I3 and I4, the subroutes remain intact. As the order of nodes is not changed, it is guaranteed that the pickup is always positioned before the dropoff. Hence, this check is not necessary for I3 and I4.

- **Constraint 4.** Constraint **4.** requires a simple check for operators I2, I3, and I4. In I1, this check is not required since nodes are not moved to other routes. In the other operators, the feasibility table presented in Table 4.1 is utilized to determine if moving the nodes is feasible for all considered nodes.

- **Constraint 5.** Constraint **5.** does not require additional checks, as it is implicitly taken care of, with the earliest departure of the inbound always being later than the latest arrival of the outbound.

Constraint	Operator			
	I1	I2	I3	I4
1				
2				
3				
4				
5				

Table 6.1: Easy constraints to be checked per operator

Checks for Constraint **3.** in I1, and checks for Constraints **6.**, **7.**, **8.**, and **9.** in all operators are discussed in the next section.

6.5.3 Updates and More Complex Constraint Checks

This section addresses the checks that require updates and discusses these updates. These include the checks for time window constraints, capacity constraints, and depot constraints as discussed in Sections 4.2.3, 4.2.2, and 4.2.4. Unlike the simple checks discussed in Section 6.5.2, the more complex constraints need to be checked for all operators. Similarly to the previous section, checks are performed for both the extraction and insertion phase when applying I2 or I3. Furthermore, note that insertion in the initial solution heuristic, shown in Algorithm 1 and discussed in Section 6.4, is rather similar to insertion in I2 and I3. Therefore, the complex constraint checks for inserting in this heuristic are discussed here as well.

Updating Order of Nodes The first update involves updating the order of nodes in the route. Once the node order is adjusted, the feasibility of Constraint **3.** in operator I1 can be evaluated by looping through the route. Simultaneously, Constraints **6.** is verified. The for loop that iterates through the route checks if the vehicle's capacity is not exceeded. Constraint **6.** is verified in I2 and I3 in this manner as well for both the extract and the insert part. These constraints are also satisfied for the insertion in the initial solution heuristic in this manner. Although in I4 the tail is independent and thus feasible in the old route, it could be the case that the other route has a smaller capacity. As a result, there is a possibility that the tail can not be placed in the other route. Therefore, I4 also requires a check for Constraint **6.**.

Updating Event Times The next update involves updating the event times. As discussed in Section 4.2.3, a route can only be feasible if all event times of the route fall within their time windows. We also showed in Figure 4.1 in this section how event times are calculated. Once all the event times are updated, each node

in the route is checked to ensure that its event time falls within the corresponding time window. If this condition is violated for any node, the route is considered infeasible, thus addressing Constraint **7.**. It also checked if the event time of the dropoff node minus the event time of the corresponding pick node does not exceed its maximum ride time, addressing Constraint **8.**. This process is similar for all operators, including the insertion in the initial solution.

Updating Current Occupation Matrix For evaluating Constraint **9.**, it is important to keep track of how many vehicles of each type are available at pool locations and when these numbers change. This information is recorded in the Current Occupation Matrix (COM). For each original route, the first and last nodes are presented in the COM. The first node indicates when a vehicle's schedule begins and, therefore, signifies a reduction of one available vehicle of this type at its pool location. The last node indicates when the vehicle completes its schedule and, therefore, signifies an increase of one available vehicle of this type at its ending pool location. The COM maintains a **chronological** record of event times of the first pickup and last dropoff events per type at each pool location, along with the current number of available vehicles at these locations per type.

The final constraint check, which is computationally expensive, involves updating this COM. When an operator modifies the first or last node of a route, the COM needs to be updated at the corresponding pool location and vehicle type entry, and its feasibility is checked. The event corresponding to the node that was previously the first or last in the route is removed from the COM. If a pickup is removed, all subsequent events in the COM, which occur later in time because of the chronological ordering, are increased by 1 - as there is an increase in available vehicles of the corresponding type at the corresponding pool location. If a dropoff is removed, all subsequent events are decreased by 1. The reverse is performed for the new first or last node. If a pickup is inserted at the start of a route, all subsequent events in the COM are decreased with 1, while for a dropoff at the end position, they are increased by 1. Subsequently, it is checked if the occupation at any point in time becomes less than 0. If this condition is the case, Constraint **9.** is violated, making the route infeasible.

It is also possible that the modification of routes changes the event time of the last node, which could have influence on the COM as well. It could for example be the case that the event in the COM that corresponds to the last node should be shifted forward, such that it is not feasible because a pickup that would use this vehicle is now earlier in time than the dropoff of the vehicle. Therefore, this should be checked as well for all operators, including I4.

Chapter 7

Methodology

In this section, we present the methodology. To begin with, we introduce three additional algorithms alongside `CombinedApproach`. The purpose of these algorithms is to facilitate the evaluation of `CombinedApproach` in Section 8. Subsequently, in Section 7.2, we discuss the reduction of available vehicles during the experiments. Furthermore, in Sections 7.3, 7.4, 7.5, and 7.6, we discuss choices made for values of hyperparameters in the objective function, neighbourhood operators, SA, and Iterated Local Search, respectively.

7.1 Three Comparative Algorithms

To assess the performance of the `CombinedApproach` algorithm described in Section 6, it is important to compare it with other approaches. Specifically, we compare the algorithm with three other algorithms, presented in Table 7.1, to observe the effects of ride-sharing, unbound vehicles, and their combination. The first comparative algorithm we examine, `NaiveApproach`, mimics the naive manual scheduling approach currently in use. This approach does neither implement ride-sharing nor independence of vehicles. The second comparative algorithm, `UnboundVehicles`, does not allow ride-sharing but allows vehicles to be not returned to their pool location. On the other hand, the third comparative algorithm, `RideSharingOnly`, incorporates ride-sharing while restricting vehicles to their pool location. Note that only in `UnboundVehicles` and `CombinedApproach` single rides are possible. We describe these three algorithms in more detail in Sections 7.1.1, 7.1.2, and 7.1.3. These algorithms are similar to `CombinedApproach`, but have some minor changes in their constraints, as we will shortly discuss. Due to their distinct constraints and objectives, modifications to the initial solution heuristic of `CombinedApproach`, shown in Algorithm 1 in Section 6.4, are discussed as well.

	Ride-sharing	
Unbound vehicles	**No**	**Yes**
No	`NaiveApproach`	`RideSharingOnly`
Yes	`UnboundVehicles`	`CombinedApproach`

Table 7.1: Overview of the four algorithms used in the experiments, indicating their binary values for ride-sharing capability and whether vehicles are constrained to pool locations.

7.1.1 Naive Approach

In the `NaiveApproach` we mimic the current manual planning process, as described in Section 3.1. The pseudocode is shown in Algorithm 5. In this algorithm, individuals can utilize the same vehicle, but serially rather than through ride-sharing. Additionally, the pool location that the vehicle belongs to is bound, and single rides are not possible. Inbound rides are always scheduled immediately after outbound rides, and outbound rides cannot occur independently or in a different vehicle. Vehicles can be assigned to a user for multiple days. Unlike `CombinedApproach`, where the total number of deviations per vehicle type per pool location always adds up to 0 at the end of the day, it is not necessary for that to be the case here. It is possible for someone to reserve a car for 2 or 3 days, and therefore that the total deviation is smaller than 0 as vehicles are 'missing'.

As described in Section 3.1, a *first come first serve* strategy is maintained. Therefore, we schedule requests according to their request date, described in Section 5.2.1. During the placement process, all routes are examined to determine suitable positions for all four nodes that correspond to the outbound and inbound events. These nodes can be placed either in an empty route or within a non-empty route where the new route remains feasible with respect to all earlier discussed constraints in Section 4.2. Besides, as ride-sharing is not allowed, Constraint **6.** is extended as follows:

10. Maximum occupation
To make sure that no rides are shared, the occupation of a vehicle should not only be smaller than the capacity of the vehicle, but also be equal to the number of users in the request. By doing this, it can be ensured that there are never multiple users from different requests present in a vehicle at the same time.

Furthermore, there is a transitive relation between the scores assigned to placing the nodes in an original empty route, placing the nodes in a non-empty original

route and placing the nodes in an additional route. Placing nodes in an original non-empty route is better than placing nodes in an original empty route to optimize vehicle utilization and obtain a more accurate assessment of the number of empty vehicles. Besides, placement in an original empty route is better than placement in an additional route. If no feasible option is available, the request is not planned. After this placement, unlike the other three algorithms, no local search is performed.

Algorithm 5 `NaiveApproach`

```
1: Initialize empty routes
2: for request in requests do
3:     PossiblePlacements = GetPossiblePlacements(outbound, inbound)
4:     if not PlacedInBestPosition(PossiblePlacements) then
5:         AddAdditional(outbound, inbound)
6:     end if
7: end for
```

7.1.2 Unbound Vehicles without Ride-sharing

In the following comparative algorithm, `UnboundVehicles`, ride-sharing is not implemented, but vehicles are not restricted to pool locations. As a result, when using this algorithm, users are not bound to their own reserved vehicle, as it is possible to separate the outbound and inbound rides and have them placed in separate routes. This means that for non-mandatory routes, the outbound ride can be scheduled without the inbound ride taking place. Besides, single rides are possible. To make sure no ride-sharing is performed, we use Constraint **10.**, which is an extension of Constraint **6.**, as previously described in Section 7.1.1. As an initial solution heuristic, we use Algorithm 1 from Section 6.4 with Constraint **10.**. The subsequent local search is the same as the one used in `CombinedApproach`, with the only difference using this extension.

7.1.3 Ride-sharing with Bound Vehicles

In the third comparative algorithm, `RideSharingOnly`, ride-sharing is allowed, but vehicles must always return to their pool locations. In this scenario, it is also possible to separate outbound and inbound rides and place them in different routes. For non-mandatory return requests, the outbound ride can be scheduled without requiring the inbound ride to be scheduled. Note that this is possible as the vehicle can be returned to its own pool location by another user. Although in fact it would be possible to include single rides in this algorithm, as they could be scheduled through ride-sharing, we do not consider them because single rides without ride-sharing are not possible in this approach. For generating an initial solution, a similar

heuristic to `CombinedApproach` is used. The pseudocode is provided in Algorithm 6. Furthermore, to ensure that vehicles always return to their pool locations, an

Algorithm 6 `RideSharingOnly` Initial Heuristic

```
Initialize empty routes
for mandatory in requests do
    PossiblePlacements = GetPossiblePlacements(outbound, inbound)
    if not PlacedInBestPosition(PossiblePlacements) then
        AddAdditional(outbound, inbound)
    end if
end for

for non-mandatory in requests do
    PossiblePlacements = GetPossiblePlacements(outbound, inbound)
    if not PlacedInBestPosition(PossiblePlacements) then
        PossiblePlacements = GetPossiblePlacements(outbound)
        if not PlacedInBestPosition(PossiblePlacements) then
            AddAdditional(outbound, inbound)
        else
            PossiblePlacements = GetPossiblePlacements(inbound)
            if not PlacedInBestPosition(PossiblePlacements) then
                AddAdditional(inbound)
            end if
        end if
    end if
end for
```

additional constraint is introduced in Algorithm 6, given in Section 6.4, and the subsequent local search:

11. **First node's location is equal to last node's location**
    To make sure that vehicles are indeed bound to pool locations, and thus that all vehicles return to their pool location, the location that corresponds to the first node should always be the same as the location that corresponds to the last node. After each neighbourhood operation, this constraint is checked for in `RideSharingOnly`.

Note the similarity with the initial solution heuristic for `CombinedApproach`, but with the difference that in Algorithm 6 single ride requests are not handled as they can not occur. In both heuristics, first the mandatory-cases are handled, in exactly the same way. For non-mandatory return requests, a similar but slightly different approach is followed. We first assess if these nodes can be placed as a whole either before or after an existing route. This is a difference with the initial solution heuristic for `CombinedApproach`. The reason for not immediately separating the outbound and inbound rides, but first looking at the possibility of

placing it as a whole before or after an existing route, is due to Constraint **11.**. Placing the outbound ride first after or before a route, followed by the inbound ride in the next step, would often result in a violation due to the change in the vehicle's final or respectively start pool location, making it not possible to be chosen as best placement. Even if the inbound ride could be placed there in the next step, returning the vehicle to the actual pool location, the violation would still occur, preventing it from happening. By placing all four nodes together at once, this issue is avoided. If this is not feasible, we separate the inbound and outbound rides. First, we examine if the outbound ride can be inserted at any position. If it cannot, all four nodes are placed in a new additional route. If it is possible, the inbound ride is also placed in the best possible position within an existing route, or alternatively in a new additional route if not feasible.

The subsequent local search is identical to the local search used in the `CombinedApproach`, with the only difference being the addition of Constraint **11.** which ensures that vehicles are bound to their pool location.

7.1.4 Returns with Bounds on Different Days

As previously mentioned, there are requests in which the outbound and inbound rides occur on different days. As described in Section 6.4, these can be treated as single rides which have to adhere to Constraint **4.**. The four algorithms handle these requests in different ways, which are outlined below:

- In `NaiveApproach`, users can reserve a car for multiple days, allowing the vehicle to be absent from its pool location as it is being used by a user.
- In the other algorithms, users cannot reserve a car for multiple days, and all vehicles must return to their pool location at the end of each day. Therefore, in both `UnboundVehicles` and `CombinedApproach`, the rules described in Constraint **4.** apply. The inbound of mandatory returns is not guaranteed because both the outbound and inbound can only occur through a ride-sharing arrangement, and it cannot be guaranteed this is feasible.
- In `RideSharingOnly`, requests with bounds on different days are never scheduled. The reason for this is they are treated as two separate single rides, and single rides are not considered in this algorithm.

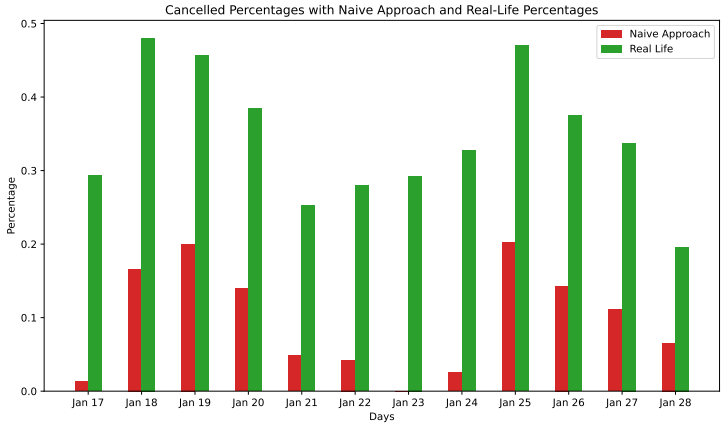

Figure 7.1: Percentage in decimal form of requests cancelled with Naive Approach and real-life scenario

7.2 Reducing Number of Vehicles

In this section, we will elaborate on the reduction of number of available vehicles used in the experiments in Section 8. This reduction aims to create a scenario that aligns more closely with real-world conditions. We will also elaborate on how we determine the percentage by which the reduction will be implemented. Figure 7.1 presents the percentage in decimal form of rejected requests for each day from Monday January 17, to Friday January 28 in real-life, and when `NaiveApproach` is run. It is evident that the naive approach, which mimics the manual planning process, yields significantly higher acceptance rates, as the cancelled percentages are lower. There are several possible reasons for this outcome, which are listed below.

- One possibility for this outcome, as described in Section 5.2.2, is the removal of irrelevant rows in the dataset.
- Another reason could be uncertainties such as traffic congestion and disruptions in reality, leading to fewer scheduled rides.
- Furthermore, a number of vehicles will be absent each day due to repair reasons.
- Besides, it is possible that in practice, they do not immediately schedule a ride when a vehicle should become available but instead wait for a few hours to ensure the vehicle's return.
- Another reason is that a number of rides are left out, as they are transformed to single rides and these can not be planned by `NaiveApproach`, as only returns are possible.

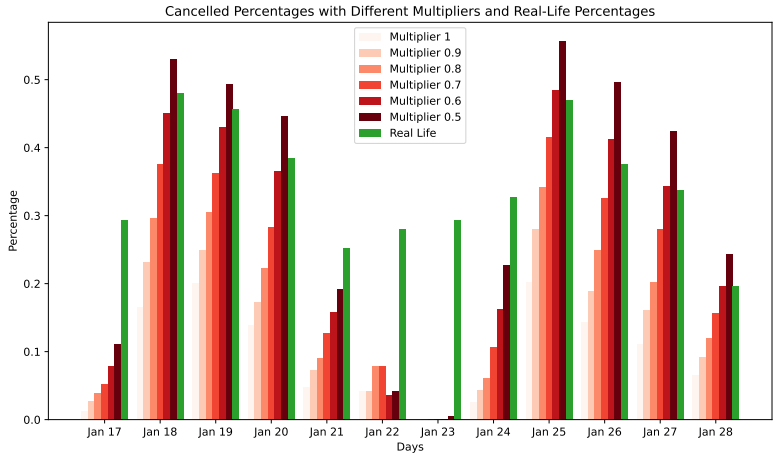

Figure 7.2: Percentage in decimal form of requests cancelled with Naive Approach with different percentages of vehicles, and real-life scenario

- One final reason is that there could be vehicles that are away from the pool location for an extended period, which is not accounted for in this analysis. We are considering data from January 17, as described earlier, and any trips that departed before that date and have not returned on the specific day are not included.

To make our model better align with reality, the number of available vehicles is reduced. Figure 7.2 illustrates a similar plot to Figure 7.1, displaying the number of rejected requests per day but with using respectively 90%, 80%, 70%, 60%, and 50% of the available cars. As expected, this figure demonstrates that as the number of available vehicles decreases, a higher percentage of requests are cancelled.

We will examine the different scenarios to determine the closest approximation to the real situation. We can already exclude the situation where 50% of the vehicles are available, as in this situation the percentage of cancelled vehicles is on some days higher than the percentage of planned vehicles as can be seen in Figure 7.2, which is not similar to the real life situation. Visually, the situation where only 60% of the vehicles are available seems to provide the closest approximation to the real life situation. To quantify this, the Euclidean distance is computed between the list of actual scheduled percentages and the list of scheduled percentages using the `NaiveApproach` with the aforementioned available percentages of vehicles. The results are presented in Table 7.2. As shown in this table, the Euclidean distance is indeed the smallest for the situation where only 60% of the vehicles are available. Therefore, from now on, the available number of all types of vehicles at all pool locations are multiplied by 0.6 and rounded to whole integers.

Percentage available vehicles	Euclidean Distance
100%	0.879
90%	0.766
80%	0.653
70%	0.538
60%	0.480

Table 7.2: Euclidean distance between the list of actual scheduled percentages and the list of scheduled percentages using `NaiveApproach` with different percentages of available vehicles

7.3 Objective Hyperparameters

As discussed in Section 4.1, the objective function comprises four components: the ratio of the total distance travelled to the total distance travelled without ride-sharing, the number of non-empty vehicles, the number of persons that cannot be accommodated within existing vehicles, and the deviation between the actual and desirable numbers of vehicles remaining at the pool locations by the end of the day. The hyperparameters α, β, γ, and δ represent the weights assigned to the respective components of the objective function. In this section, we demonstrate the influence of these hyperparameters on the quality of the solution and the importance of achieving a balanced composition of values.

Influence of α and γ To determine balanced values for α and γ, it is important to consider the trade-off between the objective component that represents the ratio of actual distance travelled to distance without ride-sharing and the number of scheduled users. This trade-off is evident in Figure 7.3. In this figure, δ is set to 2, β is set to 1, γ is set to 20, and α is varied. In the figure, the x-axis represents the value of α, while the y-axis represents the percentage of scheduled rides and the ratio of total distance travelled to distance travelled without ride-sharing. The ratio should be minimized, whereas the number of planned users should be maximized. It can be observed that when α is set to 0, the ratio is 0.93. As α increases, this ratio decreases. The percentage of scheduled users also decreases after a value of 50,000, indicating the trade-off point. However, before reaching this trade-off point, the percentage of scheduled users actually increases. This suggests that these two components support each other until a value of α is reached, where they start to impede each other. This observation is not surprising as increased ride-sharing leads to more users getting a ride, until it becomes unbalanced and the focus is overly placed on the ratio. Higher values than $\alpha = 50,000$ would prioritize ride-sharing excessively, whereas the priority lies in the number of scheduled users.

Thus, a value for α of 50,000 seems suitable in combination with $\delta = 2$, $\beta = 1$, and $\gamma = 20$ - note that we will also consider the influence of β and δ in the following

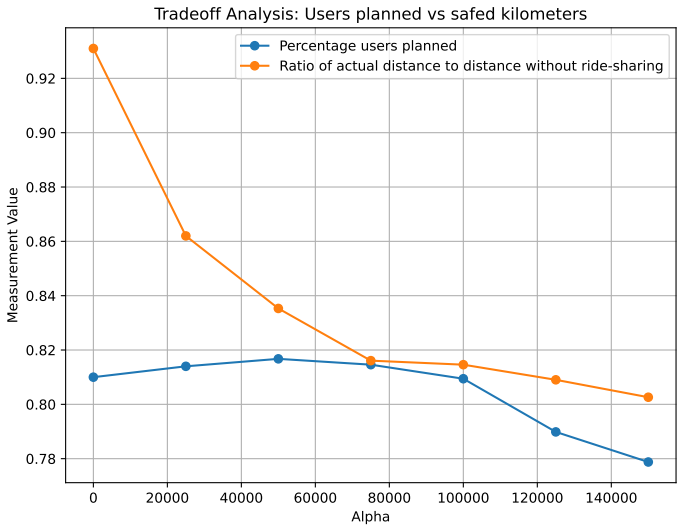

Figure 7.3: Tradeoff between the percentage of users planned and the ratio of the total distance travelled to the total distance travelled without ride-sharing

paragraphs. Therefore, we will use a value of $\alpha = 50,000$ and $\gamma = 20$ in the following sections. However, as January 18th represents the busiest day, as depicted in Figure 5.6 in Section 5.2.3, this choice might not be optimal for other days; when there are fewer departures, rides have a greater impact on the ratio. Nevertheless, further experimentation indicates that a value of 50,000 yields the desired effect consistently across all days.

Influence of β The component related to empty vehicles, with hyperparameter β, does not necessarily need to be minimized. It is not crucial to have many empty vehicles, particularly considering the trade-off with the priority of accommodating requests. However, without this component, many requests that could have been placed serially in the same vehicle are placed in separate vehicles. It is interesting to examine the number of redundant vehicles. Therefore, it is prioritized to use the same vehicle for separate requests rather than using different vehicles. Since this is a matter of priority rather than a specific goal, β does not need to be large, and a suitable value is 1. In Figure 7.4 and Figure 7.5, we observe the difference in percentages of empty and non-empty vehicles when setting β respectively to 1 and 0 while keeping the other hyperparameters constant. These results were obtained by applying a relatively small number of iterations, i.e. 6,560,000, of SA on January 18th. In this case, the percentage of empty routes was 36%, whereas with the value set to 0, it is only 8%. This clearly demonstrates the relevance of hyperparameter β even at a low value of 1.

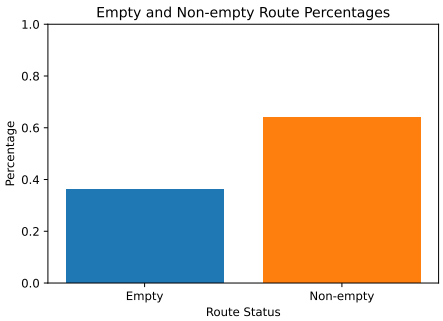

Figure 7.4: Percentage of empty vehicles ($\beta = 1$)

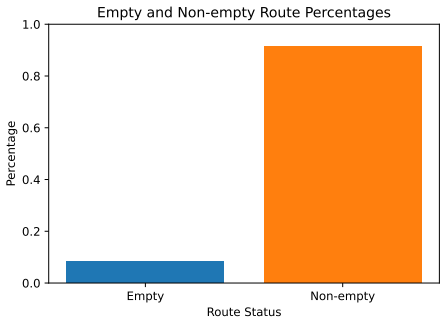

Figure 7.5: Percentage of empty vehicles ($\beta = 0$)

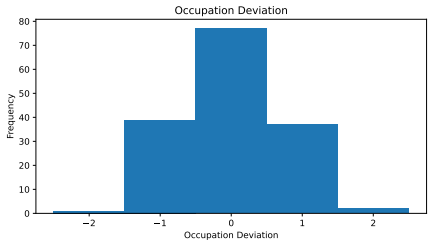

Figure 7.6: Distribution of occupation deviation for all pool locations ($\delta = 2$)

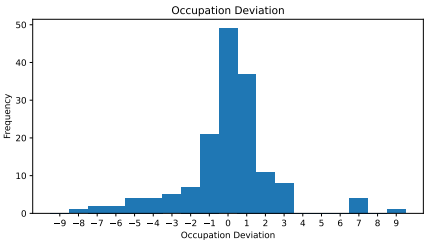

Figure 7.7: Distribution of occupation deviation for all pool locations ($\delta = 0$)

Influence of δ Defence has indicated that occupation deviation is allowed to 'breathe', meaning deviations around 1, 2, or 3 are acceptable. It has been found that a good value for δ is 2. Although this may seem small, the exponential function results in significant penalties for larger deviations. To show this, its value is set to 0 while keeping the other hyperparameter values unchanged and analyse the differences in results when running the same data and settings as before. When we examine Figure 7.6 in which $\delta = 2$, significant differences are evident compared to when its value is set to 0, shown in Figure 7.7. With a value of 2, the most common deviation is 0 with a frequency of over 75, but with a value of 0 it occurs only about 50 times. Moreover, larger deviations are observed. With $\delta = 2$, the maximum deviation was 2, occurring rarely, whereas with the value set to 0, we observe deviations as large as 9. It is evident a small value for hyperparameter δ is sufficient to prevent these deviations due to the nature of the exponential function.

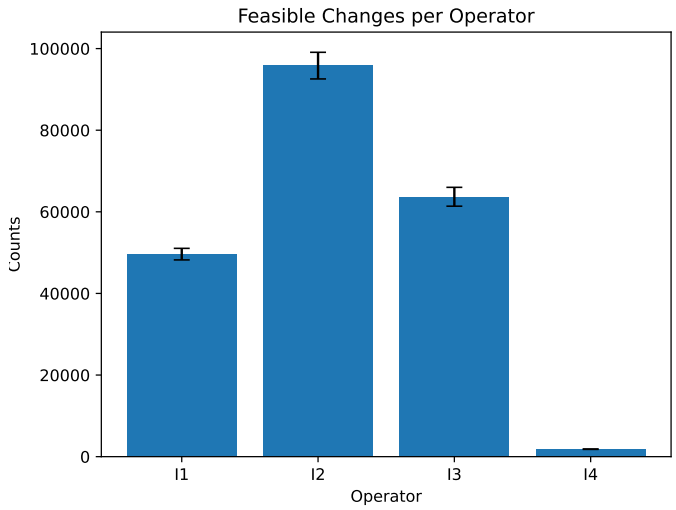

Figure 7.8: Average counts of feasible changes per operator for 5 runs

7.4 Operator Hyperparameters

To determine the probabilities at which operators I1, I2, I3, and I4 should be applied, three aspects are considered: the number of times a feasible solution is obtained after applying an operator, the number of times a new best solution is found after applying an operator, and the degree of improvement per operator.

Operators and Feasible Changes First, we examine the operators in terms of how often they create feasible changes. Figure 7.8 displays the average respective counts and standard deviations for 5 runs, with hyperparameter settings on the typical day of January 18th with values $\alpha = 50,000$, $\beta = 1$, $\gamma = 20$, $\delta = 2$ and 6,560,000 iterations with SA. Furthermore, all operators are applied with a chance of 25%. As observed in this figure, operator I2 creates on average the most feasible changes: 95826. I3 and I1 obtain a relatively lower but substantial number of feasible changes: respectively 63,690 and 49,630. Operator I4, on the other hand, leads to a very small number of feasible changes, with a count of only 1834. As we can see from the error-bars, the counts for all runs were rather close.

Operators and Improvements Next, we consider the number of improvements of the best found fitness achieved by the operators. Figure 7.9 illustrates that the majority of improvements are caused by I2, which is not surprising given its high number of feasible changes observed in Figure 7.8. Following I2, I3 exhibits a smaller number of improvements. Similarly, the low improvement count of I4 is

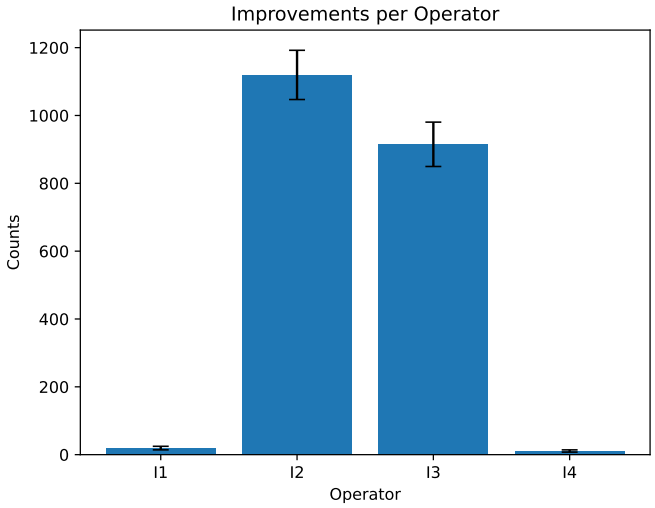

Figure 7.9: Average counts of improvements of the best found fitness per operator for 5 runs

expected due to its low feasible change count. However, if we look at I1, we see the improvement count is very low, while the feasible change count, presented in Figure 7.8 is rather high. To show the relation between the number of feasible changes and improvements, Figure 7.10 is presented. This plot is formed based on the averages of feasible change counts and improvement counts for each operator of the 5 runs. In this bar plot, each bar represents the ratio of improvement counts to feasible change counts for each operator. A higher ratio suggests a higher proportion of improvements relative to feasible changes. Based on this figure, it can be observed that I3 has the highest ratio of improvements to feasible changes among the operators. This indicates that a relatively larger percentage of the feasible changes made by I3 result in improvements. I2 follows I3 in terms of this ratio. Additionally, despite having a small absolute count of feasible changes and improvements, I4 still shows a considerable percentage of feasible changes leading to improvements. As discussed earlier, we can see that I1 has a significantly low ratio. This suggests that only a very small portion of the feasible changes made by I1 result in an improvement in the solution.

Degree of Improvement In Figure 7.11, the difference in fitness for feasible changes are depicted in a histogram. As we are minimizing, the positive side represents worsening in fitness. Note that the majority of feasible changes does not result in an improvement, as indicated by the higher frequencies on the positive x-axis, but results in a worse solution. In this paragraph, our focus is on the negative x-axis of the histogram to examine the degree of improvement per operator. We see that

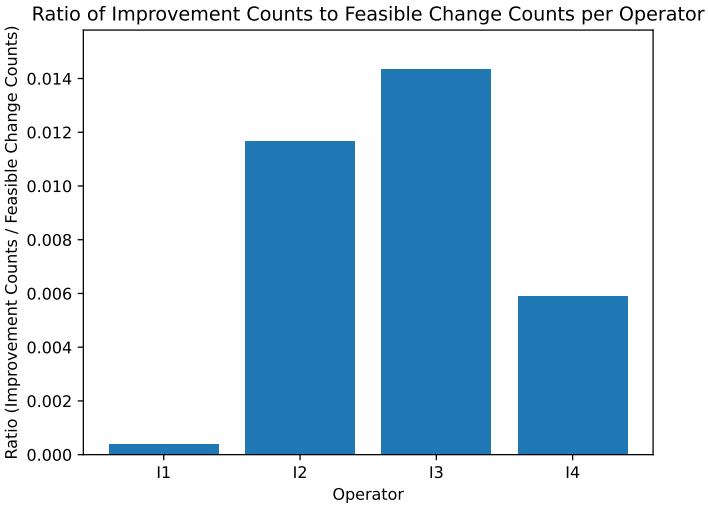

Figure 7.10: Ratio of improvement counts to feasible change counts for each operator per operator for 5 runs

only a very small portion of the changes result in improvements. Note that this number is bigger than the total number of improvements of the best found fitness, depicted in Figure 7.9. The reason for this is that in Figure 7.9 we look at improvements of the best found fitness, while in Figure 7.11 we look at improvements of the current solution's fitness. Trivially, the current solution's fitness is more often improved than the best found fitness.

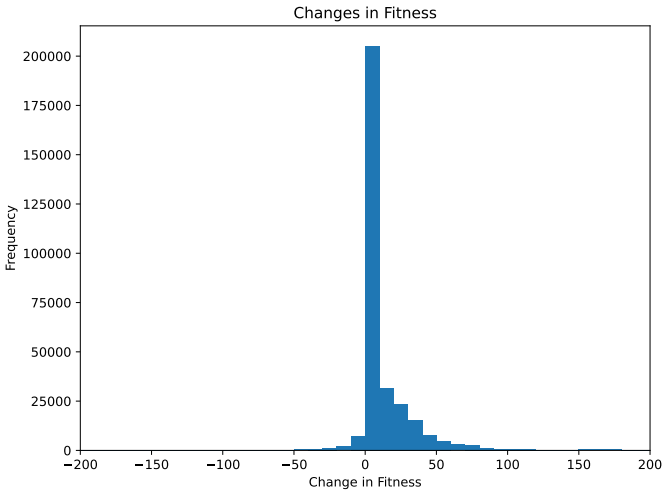

Figure 7.11: Changes in fitness for feasible changes

To evaluate the extent of improvement of the best found fitness achieved by each

Operator	Mean	Std.	Median	Maximum
I1	2.11	2.89	0.069	6.19
I2	7.04	14.61	1.00	163.37
I3	5.24	12.96	1.00	141.42
I4	5.99	0.17	5.90	6.34

Table 7.3: Summary statistics of the performance of I1, I2, I3, and I4. The statistics include the mean, standard deviation, median, and maximum values for each operator's improvement in fitness.

operator, Table 7.3 presents key statistical measures, including the mean improvement, standard deviation, median, and maximum values. Note that the improvements have been multiplied by -1, as the aim is to minimize the fitness function. From this table, it is evident that Operator I1 exhibits the lowest mean improvement of 2.11, while Operator I2 demonstrates the highest mean improvement of 7.04. Notably, also the standard deviation for Operator I2 is the biggest, indicating that it has an inconsistent performance. The median, a more robust measure less affected by outliers, provides more representative insights in improvement values for each operator. In our analysis, Operator I1 has a median improvement value of 0.069, while Operators I2, I3, and I4 have median values of 1.00, 1.00, and 5.90 respectively. Lastly, the maximum improvement values represent the highest level of fitness improvement obtained by each operator. Operator I1 and I4 achieve relatively low maximum improvement of respectively 6.19 and 6.34, while Operators I2, I3, attain maximum values of 163.37 and 141.34 respectively.

Operator Hyperparameter Values In summary, the following key points can be stated regarding the operators: Operator I2 consistently leads to feasible changes more frequently than the other operators, followed by I3 and I1. In contrast, I4 has a relatively lower occurrence of feasible changes. Analysing the ratio of improvement counts to feasible change counts reveals that I1 performs poorly in this regard, while both I2 and I3 exhibit good ratios. Additionally, although I4 does not often result in feasible changes, when it does, it often improves the solution. From the key statistical measures of the operators, it is evident that I1 has the lowest mean improvement of the best found fitness, but is consistent. On the other hand, I4 exhibits a high consistency with a high mean and median. I2 and I3 stand out particularly in terms of maximum improvements.

Based on these findings, the following decisions are made: I1 does not need to be applied very frequently since it does not significantly improve the solution. However, it can be utilized with a low probability to expand the search space by generating feasible changes. I2 and I3 lead to the most substantial improvements

overall, and also score highest in terms of the ratio of improvement counts to feasible change counts. Hence, they should be applied relatively often. Despite its infrequent occurrence of feasible changes, I4 leads to relatively good improvements on average when a change is feasible, making it worth occurring with a low probability. Therefore, the choice is made to apply the operators with the following probabilities: I1: 15%, I2: 35%, I3: 35%, I4: 15%.

7.5 SA Hyperparameters

Various hyperparameters need to be considered for SA, as depicted in Algorithm 4 in Section 6.5. The hyperparameters to be determined are the starting temperature T_0, the number of iterations per temperature N, the cooling scheduler c, and the stopping temperature T_{stop}. Suiting values for T_0 and T_{stop} can be found by considering the range within which the fitness of solutions varies. Figure 7.11 illustrates the difference in fitness displayed in a histogram. As observed, the majority of fitness worsening lie between values of 0 and 100. In the following paragraph, we start with setting the value of T_0 to the relatively high value of 200 and proceed to determine an appropriate value for N to see how the algorithm performs with these values. We will also consider a lower value for T_0. Furthermore, an appropriate value for T_{stop} is 0.01 and for the starting cooling scheduler c, a value of 0.975.

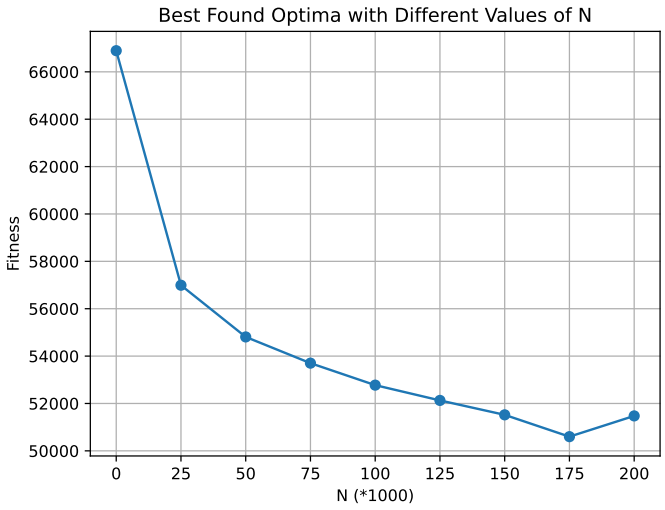

Figure 7.12: Fitness convergence with different N-values, $T_0 = 200$

Hyperparameter *N* We are interested in finding the value N at which the best found optimum converges, meaning the point where increasing N no longer has a significant effect. We run the SA algorithm with different values of N, fixing T_0

at 200, and using the previously discussed hyperparameter values for probabilities of operators and hyperparameter in the objective on the typical busy day January 18th. Figure 7.12 presents best found fitness values with different values of N, for $T_0 = 200$. As this figure shows, the best-found fitness improves in general as N increases. However, at a value of 200,000, the best-found optimum becomes worse, suggesting that higher values of N than 175,000 do not contribute to finding better solutions.

Hyperparameter T_0 and More Additional Vehicles To check if a lower value for T_0 would result in the same observed pattern, a starting temperature of 50 is also employed, presented with the orange line in Figure 7.13. This starting temperature demonstrates earlier convergence, but at a lower fitness. Therefore, a value of $T_0 = 50$ would not be appropriate.

Additionally, to make sure that the search space is not too limited due to insufficient additional vehicles for temporarily accommodating requests, we analyse the effect of adding 25% of the total number of original vehicles as additional vehicles with $T_0 = 200$. The green line in Figure 7.13 represents the best found fitnesses for different N-values with this setting. Again, a convergence is shown at a lower fitness, indicating that adding 300 additional vehicles does not lead to finding better solutions.

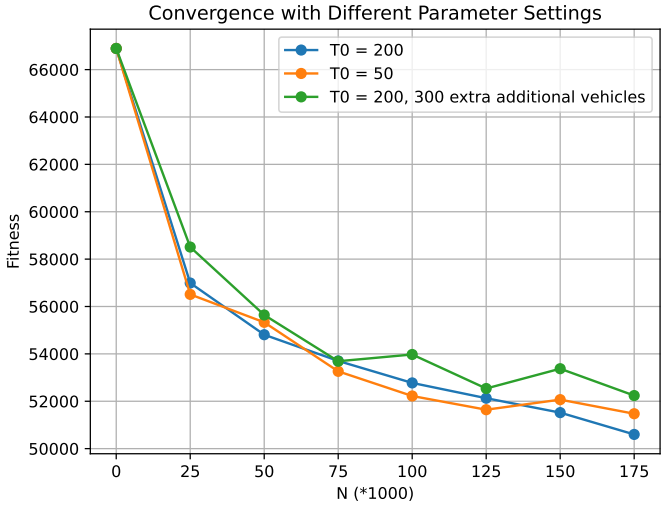

Figure 7.13: Fitness convergence with $T_0 = 200$, $T_0 = 50$, and $T_0 = 200$ with 300 extra additional vehicles

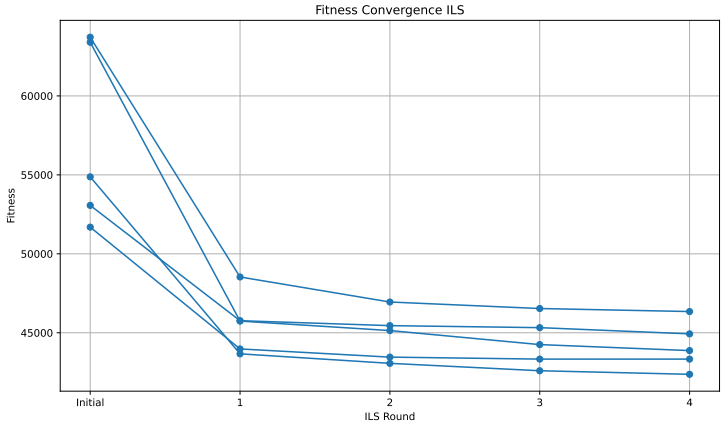

Figure 7.14: Fitness progressing after four ILS rounds with 200 perturbations

7.6 ILS Hyperparameters

To disrupt the convergence shown in Figure 7.12, an Iterated Local Search (ILS) is performed. In ILS, multiple consecutive local searches are performed, with the best-found solution from the previous local searches used as the starting point for the next one. Before each local search begins, a series of perturbations is applied to the solution. These perturbations are always accepted, even if they result in worse solutions. This process aims to let the solution escape from local optima before starting the next round of local search. It should be noted that the initial temperature hyperparameter, T_0, is reduced with each local search. As the ILS process progresses, we want less randomness as we are focussing on a smaller search space.

We perform ILS on January 17th to January 21st, with $N = 175,000$. We perform four local searches, initiating each round with 200 random perturbations and gradually decreasing the initial temperature. The first iteration begins with an initial value of 200, followed by reductions to 20, 10, and finally 5. The convergence of fitness using these settings from January 17th to January 21st is illustrated in Figure 7.14. Each blue line in the figure represents the fitness convergence for one day. In this figure, it is evident that the best-found fitness significantly decreases after the initial solution. Furthermore, the fitness for each instance continues to decrease slightly with each iteration, indicating a convergent trend. This figure demonstrates the appropriateness of the discussed ILS settings.

Chapter 8

Experiments

We perform a total of three experiments in this thesis. In the first experiment we compare the quality of the four algorithms, i.e., `NaiveApproach`, `RideSharingOnly`, `UnboundVehicles`, and `CombinedApproach`. We will elaborate more on this in Section 8.1. In the second experiment, presented in Section 8.2, we look at the effect on solutions when incorporating single rides, and how well these single rides are handled. In the third experiment, we perform a robustness analysis on the obtained solutions in Experiment 2. This experiment will be further explained in Section 8.3. All experiments are conducted using a Lenovo IdeaPad 5 Pro 16ACH6 82L500VLMH laptop. The hardware specifications of the laptop are as follows:

- **Processor** AMD Ryzen 7 5800H with Radeon Graphics, operating at a base frequency of 3.20 GHz.
- **Installed RAM** 16.0 GB, with 13.9 GB usable.
- **System Type** 64-bit operating system, x64-based processor.

8.1 Experiment 1: Comparing four Approaches

In the first experiment, we compare the quality of the four algorithms, i.e., `NaiveApproach`, `RideSharingOnly`, `UnboundVehicles`, and `CombinedApproach`. We examine the ride requests from January 17 to January 28, aiming to find the best possible solution each day. The distribution of vehicle types across pool locations is based on the previous day's end state and is used for the following day. Thus, we simulate the planning of these days. Note that in this experiment, we exclude single ride requests. The reason for this exclusion is that in `NaiveApproach` and `RideSharingOnly`, single rides are not possible since vehicles are bound to their pool location. However, in Experiment 2, described in Section 8.2, we include single ride requests.

Performance Metrics To evaluate solutions and compare the algorithms, we assess for `CombinedApproach` various aspects for each day, which are listed below:

- **Number of users in additional vehicles**: This metric represents the number of users assigned to additional vehicles. Since these additional vehicles do not correspond to real vehicles, they are not included in the scheduling. It is important to note that we consider individual users, not the number of requests, as a single request may consist of multiple users, as discussed in Section 5.2.1. This metric corresponds to the objective component with hyperparameter γ, as discussed in Section 4.1.
- **Number of users in original vehicles**: This metric represents the number of users assigned to original vehicles, i.e., the number of users that are actually scheduled. This is equal to the total number of users minus the number of users in additional vehicles.
- **Ratio of driven distance to naive distance**: This metric represents the ratio between the total distance actually driven and the distance that would be travelled without ride-sharing. It corresponds to the objective-component with hyperparameter α.
- **Average of absolute deviation**: This metric represents the average deviation per vehicle type per pool location. This measure is not in the objective, but it is closely related to the objective-component with hyperparameter δ, and is therefore insightful.
- **Empty route count**: This metric indicates the number of vehicles that are not used. This corresponds to the component in the objective with hyperparameter β.
- **Average number of rides per non-empty vehicle**: This metric represents the average number of rides assigned to a non-empty vehicle.
- **Average travel time ratio** This metric represents the average ratio of the direct travel time to the actual travel time. The actual travel time is computed as the difference between the dropoff event time and the pickup event time, minus the service time. The service time is always equal to two, as discussed in Section 5.2.2. Thus, the actual travel time encompasses waiting time, which may arise due to various reasons, such as waiting for the pickup of another person or having to take a detour to pick up someone else. Additionally, individuals can also cause waiting time for themselves. This concept becomes clear in Figure 5.5 in Section 5.2.2, where an individual departs at their earliest departure time and arrives before the start of the arrival time window.

Note that the last two metrics only consider original vehicles.

Additional Marks per Algorithm When evaluating the performance of the `NaiveApproach` algorithm, there is no need to consider the ratio between the total distance actually driven and the distance that would be travelled without ride-sharing, since this ratio is always 1.0. Additionally, we do not consider the ratio between the actual travel time and the time it would take without ride-sharing, as this ratio would also always be 1.0. Instead of focusing on the average of absolute deviation, our analysis shifts to the absence of vehicles on the following day, as some vehicles may be unavailable due to requests that exceed a duration of 1 day. Additionally, we exclude fitness evaluations from the results due to their potential to yield large values caused by the exponential component present in the objective. Moreover, the run time is fast, always requiring less than a minute to complete, and hence, it has also been excluded from the results.

When using `UnboundVehicles`, no ride-sharing takes place. Therefore, in the results, the ride-sharing ratio and travel time ratio will not be mentioned, as they will always be 1.0.

In the case of `RideSharingOnly`, the average absolute deviation of vehicles from pool locations will always be 0 because vehicles always return to their pool location. Hence, this metric will be excluded from the results for this algorithm.

Experiment Settings For each day, we first look for an initial solution, using the heuristic in Algorithm 1. Then, we perform a search with SA which is the first round of the ILS. Three more rounds of SA are performed in this ILS, using the hyperparameter settings mentioned in Table 8.1, with choices following from Sections 7.3, 7.4, 7.5, and 7.6. To monitor the progress of ILS, we also track the time taken for the total search and the optimal fitness found in each search. If the best found fitness is not improved with respect to the best found fitness in the previous round, the search is terminated.

P(I1)	15%
P(I2)	35%
P(I3)	35%
P(I4)	15%
$\boldsymbol{\alpha}$	50,000
$\boldsymbol{\beta}$	1
$\boldsymbol{\gamma}$	20
$\boldsymbol{\delta}$	2
Number of ILS rounds	4
Number of perturbations	200
$\boldsymbol{T_0}$	{200, 20, 10, 5}
$\mathbf{T_{stop}}$	0.01
$\boldsymbol{N}$	175,000
Cooling Scheduler	0.975

Table 8.1: Parameter settings

Furthermore, the quality of the initial solution varies per run. As illustrated in the initial solution heuristic, presented in Algorithm 1 in Section 6.4, the order in which ride types are scheduled is fixed. However, within a ride type, the order of placement of requests may differ. Three strategies are considered for placing requests. Strategy 1 involves randomly shuffling the requests, Strategy 2 entails sorting based on earliest departure time from smallest to largest, and Strategy 3 involves sorting based on the number of passengers from largest to smallest. In each ILS, a total of 20 initial solutions will be formed using Strategy 1, followed by an additional run of Strategies 2 and 3. The best-performing strategy will be selected as starting point for the subsequent ILS.

8.2 Experiment 2: Incorporating Single Rides

Since single rides are only possible with the `UnboundVehicles` and `CombinedApproach` algorithms, as described in Section 7.1, we focus solely on these two approaches in this experiment. The goal of this experiment is to compare the performance of these algorithms when single rides are included in the planning process, and additionally, to observe how the performance of both algorithms is affected by the inclusion of single rides in the planning. To achieve this goal, we utilize the same performance metrics as in Experiment 1 in Section 8.1, along with two additional metrics: **Number of single ride users in original vehicles** and **Number of single ride users in additional vehicles**. Furthermore, we maintain the identical experiment settings as in Experiment 1.

8.3 Experiment 3: Robustness Analysis

In this experiment, we analyse the robustness of solutions generated by the `CombinedApproach`. *Slack* holds a significant role within the context of robustness. As described in Section 2, Savelsbergh [62] (1992) introduce the notion of forward time slack for each node, indicating the extent to which the departure time of that node can be shifted forward without making the route infeasible. By scheduling the departure event time as early as feasible, as described in Section 4.2.3, we amplify this slack. The presence of slack ideally ensures that modifications in the route plan have minimal or no consequences for the feasibility of the schedule, thereby enhancing robustness. In this experiment, our focus is on investigating whether planning the departure as early as possible yields solutions robust enough to accommodate changes in a dynamic setting. More specifically, we will examine ride additions, cancellations, and uncertainty in ride times, respectively discussed in Sections 8.3.1, 8.3.2, and 8.3.3.

8.3.1 Adding Rides

In the first part of this experiment, we analyse the process of adding new rides to the solution in a dynamic setting. To achieve this, we consider the solutions obtained in Experiment 2 with the `CombinedApproach` for the days Monday, January 17th, through Friday, January 21st. Subsequently, we iteratively try to insert new ride requests to routes in the solution for five runs. These requests are actual requests originally intended for other days, and are therefore not included in the solutions. The number of rides that will be added constitutes 5% of the total number of rides for that day, while maintaining the ride type distribution outlined in Section 5.2.3 and illustrated in Figure 5.9 within this section. We assume each additional request is submitted prior to its earliest possible departure. When adding rides, we employ an insertion heuristic. This heuristic is the same as the initial solution heuristic, as presented before in Algorithm 1, except that no empty routes are initialized. Additionally, it is not only possible for mandatory returns to place their two rides as a whole either before or after a route, but also for non-mandatory returns.

It is important to note that there is no subsequent local search performed. Furthermore, note that the rides which will be added throughout the day do not influence rides that have already occurred on that day, as we assume each additional request is submitted prior to its earliest possible departure. In the results of this experiment, described in Section 9.3.1, we will provide insights in the number of rides that can be added and rides that cannot be added.

8.3.2 Removing Rides

In the second part of the experiment, we analyse removing ride requests from original routes in the same solutions as before. In five runs, a random proportion of 5% of the original ride count for that day is removed, thereby again preserving the distribution depicted in Figure 5.9 in Section 5.2.2. For removal, we employ a simple removal heuristic, shown in Algorithm 7. As before, we assume that requests for ride removal occur before the earliest possible departure. As described before, requests can exist of either two rides in the case of a return, or a single ride for a single-ride request. In either scenario, the rides are individually considered for removal. In this heuristic, we initially evaluate whether removing both nodes of a ride is feasible. If it is infeasible, we explore the potential replacement of nodes in the route to other routes. We take the pickup and dropoff to remove from the route, and subsequently all pickups that occur after this pickup and their corresponding dropoffs. This remaining route is called the 'removal route' in Algorithm 7. If the removal route is feasible, we proceed to check if the removed nodes can be replaced in other original routes in the solution using the same insertion heuristic as when adding rides to the solution, described in Section 8.3.1. If the removal route is infeasible, or if not all nodes can be replaced, the removal is infeasible. As before, this approach does not affect rides that have already taken place, as these replaced rides all occur after the removed pickup, which has also not yet begun as our assumption states. Note that the only kind of return rides considered for removal, are those with their outbound segment in the original route. For returns, both rides must be feasibly removed to achieve a successful removal. However, if the inbound ride is placed in an additional route, this ride is not removed as it is not planned to be driven anyway. In the results in Section 9.3.2, we will analyse the potential of removing rides.

8.3.3 Delays

In the final part of this experiment, we will delve into the effect of introducing delays. As discussed earlier in Section 6.1, our previous analyses have been based on the assumption of fixed travel times between locations. However, in this experiment, we aim to assess the feasibility of original routes under conditions of delay, where rides are executed at a slower pace than their nominal time. To achieve this, we will examine whether original routes remain feasible when delays occur. We must verify Constraints **7.**, **8.**, and **9.** after updating event times. The reason Constraint **9.** requires a check is that the COM might be updated due to a new event time for the last node of the route, as detailed in Section 6.5.3.

We will again utilize the solutions obtained from Experiment 2 for the same set

Algorithm 7 Removal Heuristic

```
removal_count = 0.05 × total_number_of_rides
for _ in range(removal_count) do
    Pick a random request
    for ride in range(len(request)) do
        if removal of nodes is feasible then
            continue
        else
            Try to replace nodes
            if All nodes replaced and feasible removal route then
                continue
            else
                infeasible removal
                break
            end if
        end if
    end for
    feasible removal
end for
```

of days as previously analysed. Subsequently, we will modify the travel times of rides that occur during the morning and evening rush hours, namely those with pickup event times between 6 AM and 9 AM, or between 4 PM and 7 PM, as shown in Figure 5.13 in Section 5.2.3. This adjustment will involve multiplying the nominal ride time by a multiplier. The multiplier will be determined based on a normal distribution with an average of 1.0 and a standard deviation of 0.06. However, only values within the range of 1.0 to 1.2 will be considered. If the multiplier lies outside this range, it is given a new value based on the distribution, until it lies inside the range. In Section 9.3.3, we will analyse the effects of these delays in 5 runs for each of the five days and assess their implications.

Chapter 9

Results

In this section, we present the results of Experiments 1, 2, and 3, as described respectively in Sections 8.1, 8.2, and 8.3.

9.1 Results Experiment 1: Comparing four Approaches

In this section, we present the results for Experiment 1, in which the four approaches are compared as described in Section 8.1. The results are displayed in Tables A.1, A.2, A.3, A.4, A.5, A.6, A.7, and A.8 in Appendix A, respectively for the first and second week of `NaiveApproach`, `UnboundVehicles`, `RideSharingOnly`, and `CombinedApproach`. Below, we will discuss these results per metric.

Fitness Convergence To visualize the convergence of fitness for each algorithm, we plot the fitness convergence in Figure 9.1 for all ten weekdays, starting from the fitness of the initial solution, along with the ILS rounds. Similarly as before, each line represents the fitness convergence for one day. Red lines represent `UnboundVehicles`, orange lines `RideSharingOnly`, and green lines `CombinedApproach`. Several significant observations can be made from this figure. Firstly, we notice that convergence occurs for all algorithms and instances. Moreover, it is important to observe that `CombinedApproach` exhibits a notably larger improvement after the first ILS round compared to `UnboundVehicles` and `RideSharingOnly`. However, there is still a more substantial decrease for `UnboundVehicles` than for `RideSharingOnly`. This observation indicates that the search spaces of `UnboundVehicles`, and especially `CombinedApproach`, are larger than that of `RideSharingOnly`. This is further supported by the fact that in `UnboundVehicles`, and particularly seen in `CombinedApproach`, substantial fitness improvements often occur in later rounds. On the other hand, `RideSharingOnly` generally tends to reach complete convergence. The fact that in `RideSharingOnly`

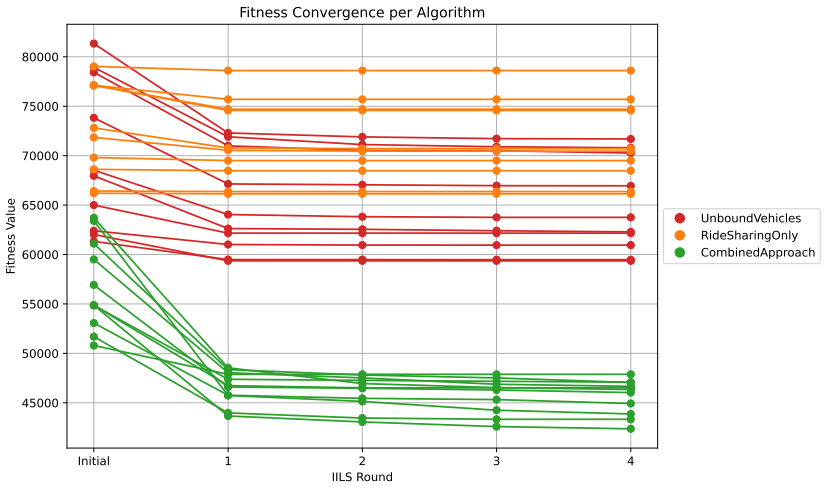

Figure 9.1: Fitness convergence per algorithm for all weekdays

a number of rides are not considered because their bounds occur on different days, as discussed in Section 7.1.4, may contribute to this.

Planned Number of Users As discussed earlier, the most important component of the objective for the Ministry of Defence is to schedule as many users as possible. Figure 9.2 illustrates, for each day and each of the four algorithms, the number of users that have been scheduled within original routes. Additionally, the total number of users on each day is also depicted.

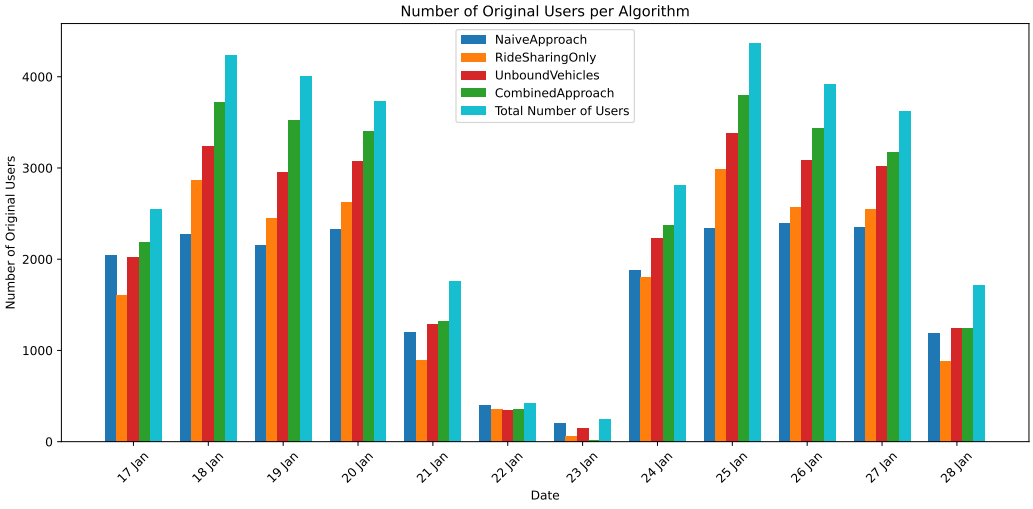

Figure 9.2: Planned number of users for all algorithms

The key observation from this figure is that, on the busy days - Tuesday, Wednesday, and Thursday — `NaiveApproach` schedules the least number of users. This is

followed, respectively, by the `RideSharingOnly` and `UnboundVehicles` algorithms. Our proposed `CombinedApproach`, on the other hand, schedules the highest number of users on all the busy days. Often, the number of scheduled users on these days falls approximately between 85% and 90%. This marks a significant improvement compared to the `NaiveApproach`, which schedules between 55% and 65% of the users on busy days.

When examining the less busy days — Monday, Friday, Saturday, and particularly Sunday — it can be observed that the difference in the number of scheduled users among the different algorithms is considerably smaller. Notably, on Sunday January 23rd, `NaiveApproach` schedules more users than all other algorithms. The reason for this phenomenon is twofold. Firstly, with fewer requests, the advantages of ride-sharing and unbound vehicles become less pronounced. This highlights that the benefits become more prominent when dealing with a larger number of requests, allowing for more combinations. Secondly, around the weekends, there are relatively more returns from which the bounds fall on different days, as discussed in Section 5.2.2. As discussed earlier, `NaiveApproach` may assign a vehicle for multiple days to a user, which is not the case for the other three algorithms. As mentioned in Section 7.1.4, `RideSharingOnly` never schedules requests with bounds on different days - which is also the reason of its bad performance on the Mondays and Fridays. For the other two algorithms, they can only be accommodated through ride-sharing. However, with few other rides available, the likelihood of finding a feasible ride-sharing arrangement is diminished.

Ratio of Driven Distance to Naive Distance Another important component of the objective is the ratio of driven distance compared to the distance that would be covered without ride-sharing. Figure 9.3 presents these ratios for the four algorithms.

For both `NaiveApproach` and `UnboundVehicles`, where ride-sharing cannot occur, the ratio is trivially always 1.0. In the case of `RideSharingOnly`, the ratio consistently falls between 0.95 and 1.0. While this signifies an advancement over the other two algorithms, it is not a huge improvement. However, upon examination of the `CombinedApproach`, a noteworthy enhancement in this ratio becomes evident. On weekdays, this ratio ranges approximately between 0.70 and 0.75. To put this into perspective, a ride-sharing ratio of 0.715 on Tuesday, January 18th, corresponds to a reduction of nearly 50,000 kilometres. Therefore, it becomes evident that the combination of ride-sharing and unbound vehicles within `CombinedApproach` not only elevates the number of planned users, as demonstrated earlier, but also amplifies the potential for ride-sharing compared to the two extensions employed independently. This phenomenon is attributed to the fact that the decoupling of vehicles

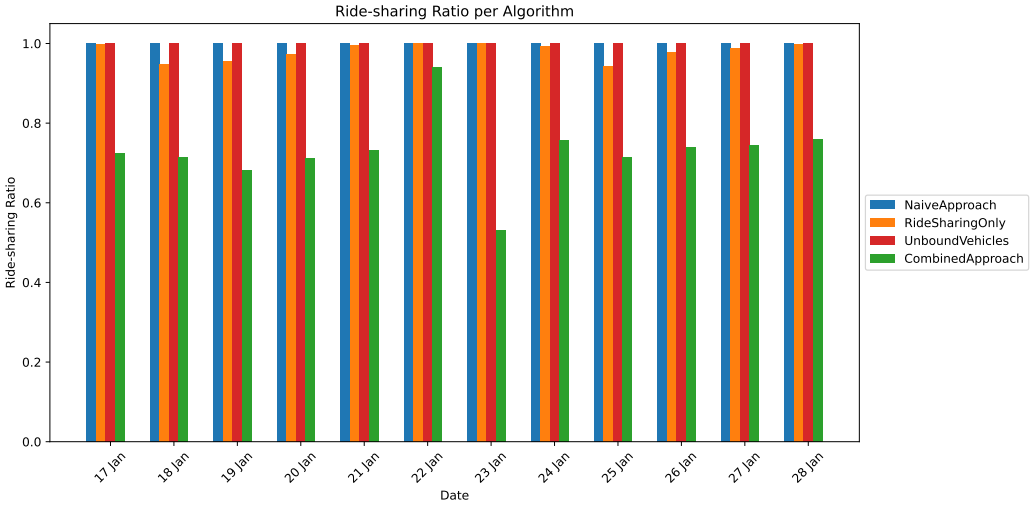

Figure 9.3: Ratio of driven distance to naive distance for all algorithms

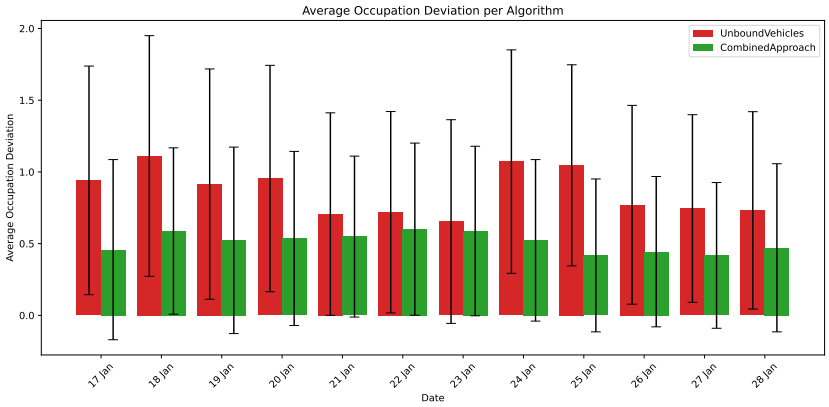

Figure 9.4: Average absolute occupation deviation for two algorithms

from pool locations leads to a substantially greater number of feasible ride-sharing arrangements.

Occupation Deviation To ensure that the number of available vehicles at the end of the day does not deviate too much from the desirable number, the occupation deviation component is, as discussed before, incorporated into the objective. Figure 9.4 presents the average absolute occupation deviation along with its standard deviation for different days, focusing on `UnboundVehicles` and `CombinedApproach`. Note that the occupation deviation is always 0 for both `NaiveApproach` and `RideSharingOnly`, and thus is not represented in the figure.

Observing the results for both algorithms, it is evident that the average absolute deviation remains notably low, never exceeding 1.12 for `UnboundVehicles` and 0.60

for `CombinedApproach`. Nevertheless, the standard deviations are relatively large, occasionally resulting in errorbars reaching 2.0, indicating that a number of absolute deviations are 2 or higher. As anticipated and discussed in Section 7.3, this figure confirms that the exponential nature of the objective component can ensure that small values for δ are sufficient to mitigate substantial deviations.

Empty Routes As discussed in Section 7.3, the inclusion of the empty routes component in the objective is not a primary goal; rather, it serves to indicate a preference for utilizing the same vehicle whenever possible, as opposed to employing a new vehicle. This approach enables us to objectively analyse, across all algorithms, the extent to which vehicles remain unused on each day. The number of empty rides for each day is depicted in Figure 9.5, alongside the total number of available vehicles.

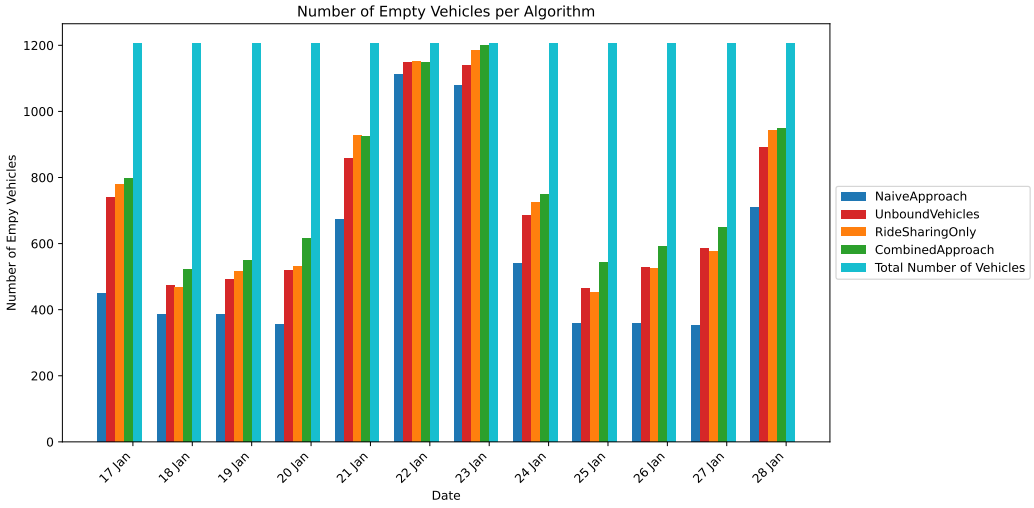

Figure 9.5: Number of empty routes for all algorithms

As evident from this figure, it is clear that a greater number of vehicles are deployed on busy days. Moreover, a notable observation is that vehicles remain unutilized on every day for all algorithms, even though not all users are scheduled, as depicted in Figure 9.2. This observation suggests that many requests originate from the same location, leading to a shortage of vehicles at that pool location while causing a surplus at other locations. However, it might also be possible that a number of mandatory return rides with bounds occurring on a different day are not able to be scheduled, as no ride-sharing arrangement can be formed. Additionally, it is noteworthy that the three more complex algorithms, particularly the `CombinedApproach`, consistently employ fewer vehicles than the `NaiveApproach`, even though in general a higher number of rides are scheduled. This implies that,

on average, the advanced algorithms accommodate more rides per vehicle than the `NaiveApproach`. In the following paragraph, we will substantiate this observation.

Number of Rides per Vehicle In Figure 9.6, the average number of rides per vehicle along with their standard deviations are depicted for the four algorithms. With `NaiveApproach`, an average of around two rides are scheduled on busy days, often corresponding to a single inbound and outbound segment of a return route. As anticipated, the three more complex algorithms indeed schedule a greater number of rides per vehicle compared to `NaiveApproach`. Specifically, it is evident that `UnboundVehicles` schedules, on average, more rides per route on all days than `RideSharingOnly`. Furthermore, `CombinedApproach` achieves an average of around four rides per route on busy days. However, the presence of substantial standard deviations indicates that routes with two rides as well as those with six rides or even more are likely to exist on busy days. This figure shows again that `CombinedApproach` results in a significantly more efficient utilization of vehicles compared to `NaiveApproach`.

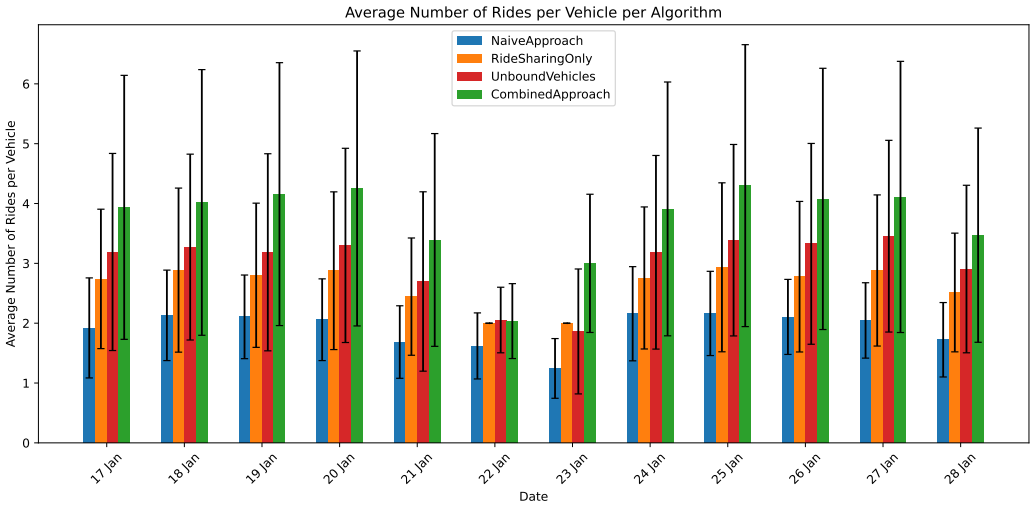

Figure 9.6: Average number of rides per vehicle for all algorithms

Travel time Ratio Finally, we examine the average ratio of direct travel time to actual travel time. Given that the maximum ride time is 1.2 times the nominal ride time, as discussed in Section 5.2.2, this value will always fall between 1.0 and 1.2. Figure 9.7 displays the travel time ratios for `RideSharingOnly` and `CombinedApproach`, as these algorithms can experience delays due to ride-sharing. As observed in this figure, the standard deviations for both algorithms do indeed lie within the range of approximately 1.0 to 1.2. Furthermore, the average ratios

exhibit minimal disparities amongst themselves, roughly hovering around 1.1.

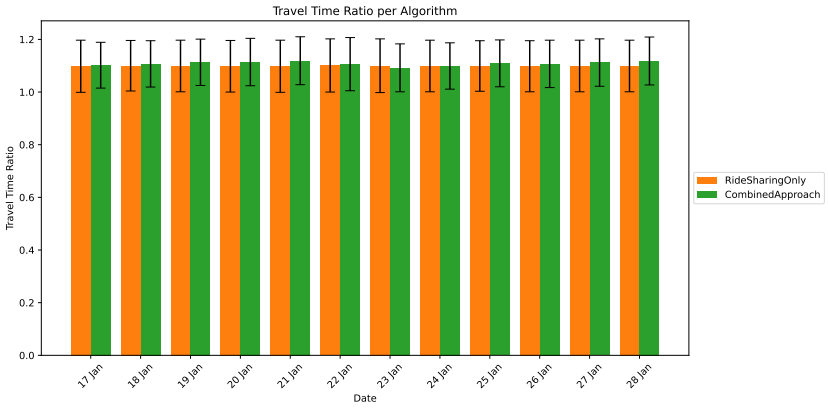

Figure 9.7: Average travel time ratio for all algorithms

9.2 Results Experiment 2: Incorporating Single Rides

In this section, we will examine the effects of incorporating single rides into the planning, as described in Section 8.2. The results are shown in Tables B.1, B.2, B.3, and B.4 in Appendix B, respectively for the first and second week of `UnboundVehicles` and `CombinedApproach`. Upon investigating the inclusion of single rides, we observe that there are no significant differences within the algorithms in the performance metrics of average absolute deviation and average number of rides per vehicle. These comparisons are illustrated in Figure C.1 and Figure C.2 in Appendix C respectively. However, we do notice a slight reduction in the counts of empty routes on busy days for both algorithms when single rides are introduced, as depicted in Figure C.3 in Appendix C. This is unsurprising given the increased number of requests.

Examining Figure 9.8, we can observe the number of scheduled single ride users for the two algorithms. Here, it becomes evident that `CombinedApproach` better handles the scheduling of single rides, and is able to schedule almost all single ride request on all days. Furthermore, Figure 9.9 compares the number of non-planned users for `UnboundVehicles` and `CombinedApproach` when single rides are allowed and when they are not. While we see an increase of non-planned users for `UnboundVehicles` on all days when single rides are included, except for January 27th, we see a decrease on multiple days for `CombinedApproach`. A reason for this is that single rides allow for a more complex search space with better solutions, in which more ride-sharing occurs. As single rides exclusively travel between pool locations, these rides are favourable for ride-sharing. Figure 9.10 supports this, as it indicates on many days a small decrease in the ratio of travel distance.

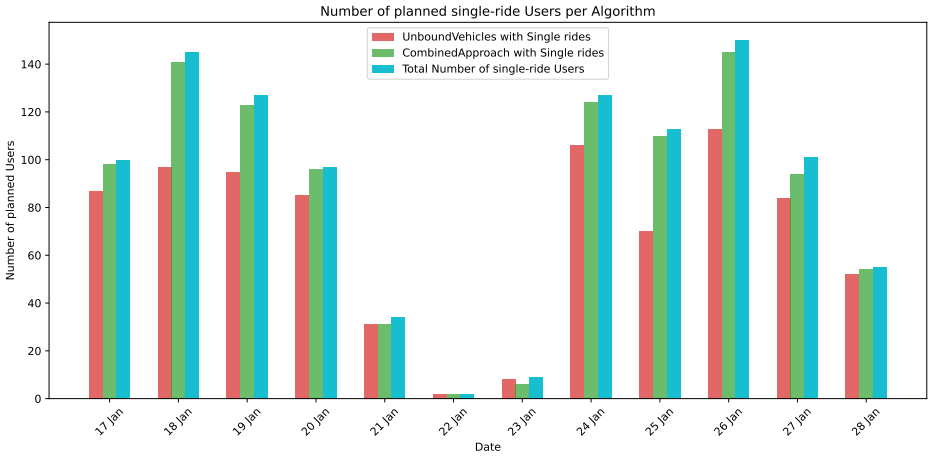

Figure 9.8: Number of single ride users planned for two algorithms

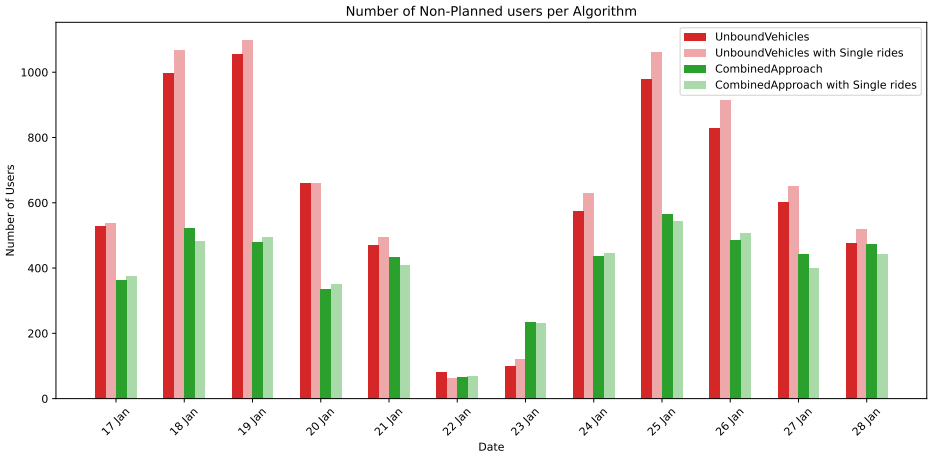

Figure 9.9: Difference in number of non-planned users with and without single rides

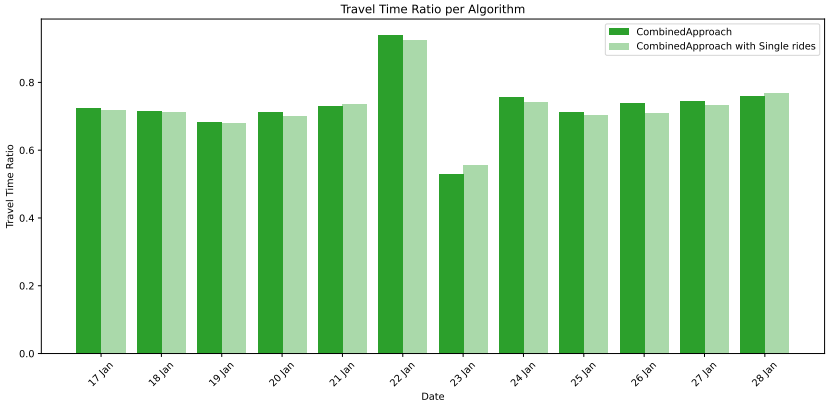

Figure 9.10: Difference in travel time ratio with and without single rides

9.3 Results Experiment 3: Robustness Analysis

In this section, we analyse the robustness of the ride-schedules, obtained in Experiment 2. More specifically, in Sections 9.3.1 and 9.3.2 we will analyse the process of respectively adding rides and removing rides from existing solutions. Furthermore, in Section 9.3.3 we will look at the effect delays have on existing schedules.

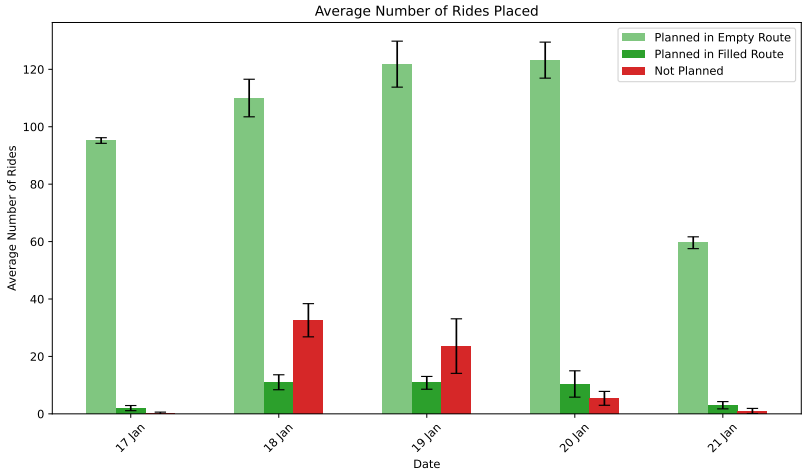

Figure 9.11: Average number of rides that can be placed and that can not be placed

9.3.1 Results Adding Rides

In this section, we observe the process of adding rides to existing solutions of Experiment 2, as described in Section 8.3.1. In Figure 9.11, the average number of rides that can be added to the existing schedule on each of the first five working days is depicted, along with the count of rides that cannot be accommodated. Among the scheduled rides, a distinction is made between placing rides in empty routes and non-empty routes. As evident from this figure, the majority of rides can be scheduled. On Tuesday 18th, which is the busiest day and therefore the most challenging for scheduling, approximately 78% of the rides are still accommodated on average. The majority of the scheduled rides are placed in empty routes. As discussed in Section 9.1, many vehicles remain unused in the schedules. As shown, these vehicles can now be efficiently employed for scheduling of new rides. Examining the removal of rides from non-empty routes and the introduction of delays to rides in non-empty routes in respectively Sections 9.3.2 and 9.3.3 will tell us more about robustness of non-empty routes.

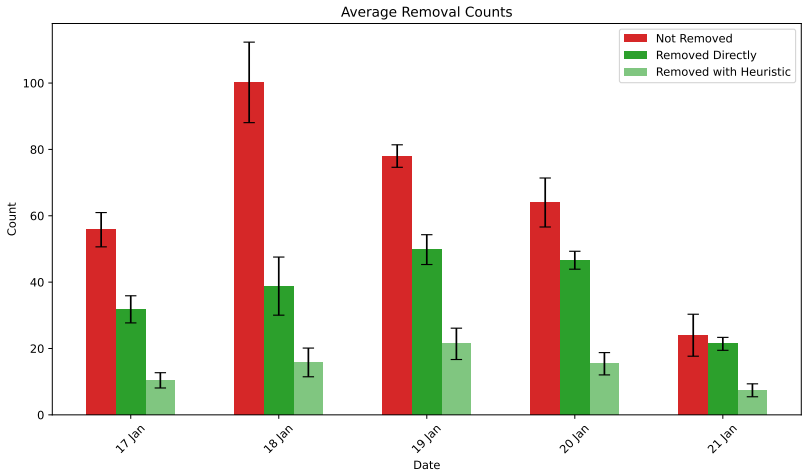

Figure 9.12: Average number of rides that can be removed and that can not be removed

9.3.2 Results Removing Rides

In this section, we will analyse the removal of rides from existing schedules, as described in Section 8.3.2. In Figure 9.12, we present the average number of rides per day that cannot be removed, can be directly removed, or can be removed using the heuristic outlined in Algorithm 7. As observed in the results of Experiment 2, detailed in Section 9.2, busy days exhibit a low distance ratio, a high average number of rides per vehicle, and a majority of users scheduled. These findings suggest complex solutions where rides are interdependent. From Figure 9.12 it becomes apparent that indeed, on busy days, the removal of rides proves to be challenging. While the majority of rides cannot be removed on average for Monday and Tuesday, nearly half of the rides on Wednesday and Thursday can be removed. On Friday, more than half of the rides can on average be removed. Additionally, the figure highlights the value of the introduced heuristic. Across all days, it is consistently able to remove a notable number of rides on average.

9.3.3 Results Delays

In this section, we will look at the consequence delays have, as described in Section 8.3.3. Figure 9.13 illustrates, for each day, the average number of feasible and infeasible routes following the introduction of delays. From this figure, it is evident that the percentage of infeasible routes lies around 23% on each day. While the majority of rides remain feasible, it is crucial to acknowledge that a portion of the routes does become infeasible due to the introduction of delays. Once again, the results show that the complex schedules leave not much room for modifications.

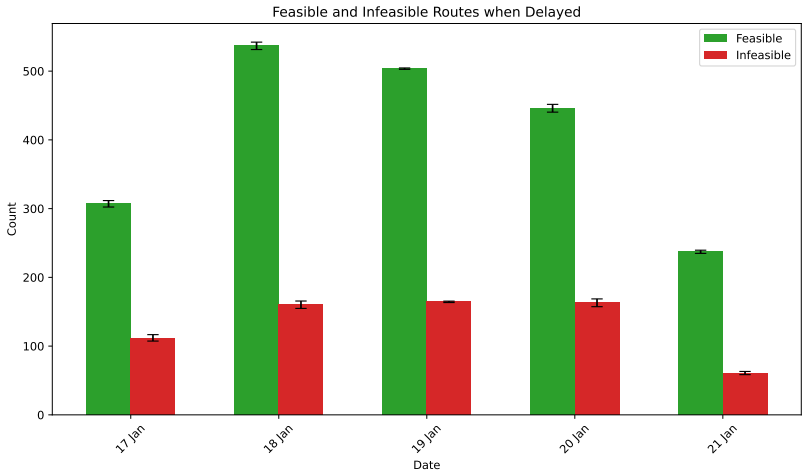

Figure 9.13: Average number of feasible and infeasible routes after delays

In Figure 9.14, we present per constraint the average number of rides with violations across five runs. Specifically, we focus only on Constraints **7.** and **8.**, as the constraint for availability of vehicles, Constraint **9.**, did not once cause infeasibility. Notably, this figure clearly illustrates that the majority of infeasibilities are caused by excessive ride times. This observation is important, as it indicates that simply increasing slack would not alleviate the issue of routes becoming unfeasible with delays. While enhanced slack could mitigate violations of Constraint **7.**, it would not address the underlying problem of excessive ride times.

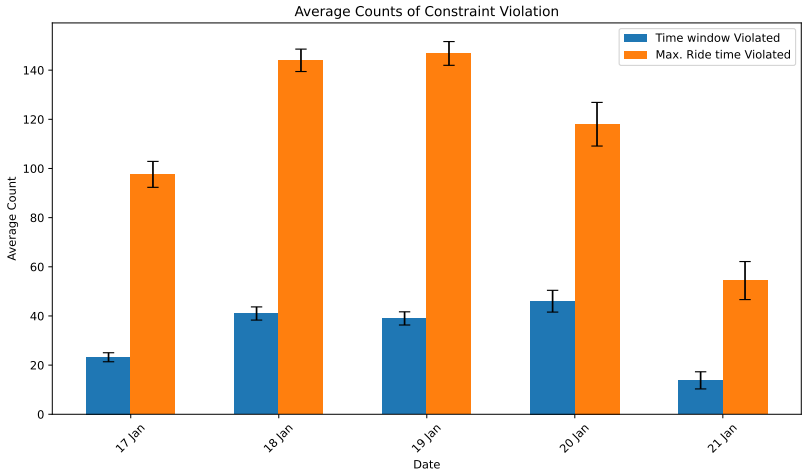

Figure 9.14: Average number of constraint violations

9.4 Conclusion of Results

The results discussed in Sections 9.1, 9.2, and 9.3 provide insights into the performances of the algorithms, their handling of single rides, and the robustness of the solutions. In this section, we will summarize these findings.

Firstly, we observed that convergence occurs for all algorithms after multiple ILS rounds. Although the convergence of `CombinedApproach` is not complete, the improvements diminish over time. This suggests that `CombinedApproach` navigates a more complex search space compared to other algorithms. Furthermore, `CombinedApproach` is the most efficient algorithm in scheduling users, and its distance ratio is significantly better than that of `RideSharingOnly`. Combining ride-sharing and vehicle decoupling results in superior solutions, outperforming either approach alone. Interestingly, `CombinedApproach` also tends to use fewer vehicles than the other algorithms, while achieving a higher average number of rides per vehicle. These advancements come with the minimal cost of a small average occupation deviation and a moderate increase in average ride time.

The handling of single rides by `CombinedApproach` is also efficient. It successfully incorporates nearly all single rides into the schedule. This is likely due to the inherent nature of single rides, which often facilitate ride-sharing arrangements. As a result, the inclusion of single rides contributes to a reduction in the distance ratio on many days, enhancing the overall efficiency of the solution.

Lastly, the addition of rides to solutions formed by `CombinedApproach` generally poses no significant issue. The presence of numerous empty routes allows for the accommodation of these rides. However, removing rides proved to be more challenging, particularly on busy days. This implies complex interdependencies among rides. Nevertheless, a portion of rides could be removed on all observed days, and the removal heuristic contributed to this process. Additionally, even though the introduction of delays during peak hours maintained the feasibility of the majority of routes, a consistent proportion of routes did become infeasible on all days. The primary reason for this infeasibility is the exceeding of maximum ride time due to ride-sharing arrangements.

Chapter 10

Discussion

In this section, we will discuss our work. We begin by addressing the limitations of our study in Section 10.1, followed by an exploration of the implications of our findings in Section 10.2. Furthermore, we provide suggestions for future research directions in Section 10.3.

10.1 Limitations

Our work is subject to several limitations due to constraints in resources and time. It is important to examine these limitations and reflect on their effects. In this section, we explore limitations across different aspects of our research. Note that some of these limitations will be further discussed in Section 10.3, when discussing future research directions.

Objective One of the areas where our research encounters limitations is in the formulation of the objective function. As outlined in Section 4.1, the objective comprises four components: the non-served users, total travel distance ratio, occupation deviation at pool locations, and the number of non-empty vehicles. In Section 9.3, we have observed a significant level of robustness in the generated schedules. However, on busy days, the removal of rides and the introduction of delays often lead to infeasibilities. Despite implementing the earliest possible departure times to increase slack, it appears insufficient to handle these changes well. As discussed in Section 9.3.3, the addition of slack does not address the issue of exceeding maximum ride times, particularly in scenarios with consecutive delays. This problem is likely caused by ride-sharing arrangements that become problematic when combined with successive delays. To further enhance the robustness, we could incorporate components in the objective that increase robustness. More details about future work in this regard can be found in Section 10.3.

Another limitation within the objective is associated with the occupation deviation component. Ideally, the absence of a vehicle should be penalized inversely proportional to the desirable vehicle count. For example, if there are only three vehicles of a particular type at a pool location, a deviation of 1 should incur a greater penalty compared to a scenario where fifty vehicles of that type are present. However, given the minor deviations in vehicle occupation for all types, as indicated in Sections 8.1 and 8.2, this limitation does not significantly impact the outcomes.

Data A big challenge in this work is the inconsistency between the available data and the input data we need for our algorithm. Consequently, preprocessing and adaptations on the data are performed, making it more uncertain how the algorithms would perform in a real-life scenario without unedited data. A first example of this data editing is the removal of rides, as discussed in Section 6.3, due to missing information. Besides, as the ride request data only provides the start and end times of reservations, time windows were formed, assuming a maximum travel time of 1.2 times the nominal travel time for each user, which might not be realistic.

Furthermore, to cluster locations, we used a string similarity algorithm, as also discussed in Section 6.3. This also brings multiple limitations with it. To start with, the algorithm may have erroneously clustered locations that should have remained separate. Instances of such locations include 'Brunssum' and 'Bussum'. Besides, a substantial number of former mandatory returns are edited and labelled and processed as non-mandatory returns. This occurs for example if rides to the non-pool location 'Utrecht' are clustered with rides to the pool location 'Kromhout Kazerne'. Consequently, the improvements our algorithm yields, shown in the results in Sections 9.1 and 9.2, might be exaggerated as there are more non-mandatory returns, therefore greater vehicle availability at pool location, facilitating easier accommodation of other rides. Clustering locations as 'Brunssum' and 'Bussum' would also contribute to this exaggeration.

The distance matrix employed for distance and travel time retrieval, as described in Section 5.3, also has its limitations. While the *Google Distance Matrix API* was used for a number of entries, estimations through regressions were adopted for others due to API request limitations. Despite the regression models being highly accurate, inconsistencies are likely to exist between actual values and our estimations.

Solution Method Our solution method is limited by a number of simplifications and assumptions. Apart from the assumption of maximum ride times, as previously discussed, other assumptions are also made that do not fully align with real-world conditions. For instance, we assume the absence of uncertainties and disturbances,

leading to each ride precisely following the duration provided by the distance matrix. In reality, peak hours and traffic conditions could impact travel times. Additionally, we do not account for fuelling, which potentially leads to delays and detours. However, results of Experiment 3, outlined in Section 9.3.3, demonstrate that small delays have limited influence on route feasibility. Furthermore, it is important to highlight that the `NaiveApproach`, mimicking the current planning system in use, managed to accommodate more requests than are planned in reality, as discussed in Section 7.2. To address this concern, we reduced the number of available vehicles, as shown in the same section. This adjustment is also inconsistent with the real-life scenario.

Another limitation in the solution method is the use of relatively restrictive neighbourhood operators within the solution method. As demonstrated in Section 7.4, operators I2 and I3 prove particularly useful and are predominantly employed. More complex operators could expand the search space, potentially leading to improved solutions, as elaborated on in Section 10.3.

Another aspect to consider is the runtime of the `CombinedApproach`. As evident in the results presented in Sections 9.1 and 9.2, the algorithm's execution time is notably slow, with a typical ILS run of four rounds potentially lasting up to four hours. While our constraint checking methods, as described in Sections 6.5.2 and 6.5.3, enhance efficiency, the algorithm's runtime is moderate. This runtime concern, however, is not a significant issue for Defence's applications, as the majority of ride requests are known a day in advance, allowing overnight algorithm execution. Besides, our instances are relative large, which also accounts for the moderate runtime, as further elaborated on in Section 10.2.

Experiments A notable limitation in our experiments, as highlighted by the results in Sections 9.1 and 9.2, is that after four ILS rounds, the best found fitness is not always completely converged, indicating that the search is not extensive enough. An example of this can be observed in the results of `CombinedApproach` on January 19th, as outlined in Section 9.1. In this case, the third round yielded a fitness of 44251, followed by 43872 in the final round, indicating a potential ongoing decline in fitness in additional rounds. It is noteworthy, however, that convergence is most of the time observed, especially on calmer days and when the comparative algorithms with narrower search spaces are employed.

Besides, it is important to note that we chose `RideSharingOnly` to not consider returns with bounds on different days, as discussed in Section 7.1.4. The reason for this is that `RideSharingOnly` does not handle single rides, and the two bounds on different days could be considered single rides. However, because a number of returns have their bound on different days, as shown in Figure 5.7 in Section 5.2.3,

corresponding users can not be planned. This might lead to worse prestation of `RideSharingOnly` in terms of users being planned, than would be the case if we did consider these returns, especially around weekend days when these type of rides occur relatively often. Future research could point out if this is indeed the case.

A final significant limitation is the absence of a global optimum for us to compare the solutions, shown in Sections 9.1 and 9.2. Despite frequently observing convergence and being able to compare our solutions to the initial solutions obtained through the simple insertion heuristic and the solution from the `NaiveApproach`, we cannot confidently assess how well the found optima are compared to the global optimum. More on how to deal with this in future research can be read in Section 10.3.

10.2 Implications

In this section, we explore the broader significance and practical implications of our research. First, we will discuss its contributions to the field, and next we discuss the real-world impact of our work.

Theoretical implications To our knowledge, this study is the first to address and solve a unique combination of the ride-sharing and car-sharing problem, in which cars are owned by a company and stationed at pool locations, and both rides and cars are shared between users. As described in Section 2, SA and variants have yielded satisfactory results in ride-sharing problems [57] [74] [12] [40]. The outcomes of this research support these findings and demonstrate that SA is effective in yielding good solutions for this variant of the ride-sharing problem. Moreover, the identified superior results when combining ride-sharing and the detachment of vehicles from their pool locations, outperforming either approach alone, constitutes a valuable contribution to the field.

Besides, the ride-sharing problem being addressed in this thesis is characterized as a broad extension of the original ride-sharing problem. Whereas most studies in literature limit their work to one or two additions, we include multiple heterogeneous vehicles and depots, strict time constraints, consideration of user preferences, limitations on vehicle capacity, and flexible roles of drivers and riders. Besides, unlike other work which often considers one component in the objective such as minimizing cost, our multi-objective function aims to minimize the number of non-served users, the ratio of the total distance travelled to that without ride-sharing, the deviation from the desirable occupation of vehicles at pool locations, and the number of used vehicles.

Lastly, it is worth noting that existing literature commonly deals with problems

on a significantly smaller scale, involving fewer requests, depots, and vehicles than our problem statement. For instance, Masmoudi, Hosny, Braekers, and Dammak [40] (2016) classify instances as large when the number of requests fall within a range of 24 to 144 requests, the number of vehicles varies from 3 to 13, while using a single depot. Furthermore, Chen, Mes, Schutten, and Quint [15] (2019) define large instances as those involving 500 to 1,000 participants. In contrast, our problem entails a higher complexity, featuring instances with over 1200 vehicles, more than 50 depots, and in excess of 3,000 requests during peak days.

Practical implications The practical impact of our research is significant, especially in guiding future planning system development for the Ministry of Defence. Despite the acknowledged limitations highlighted in Section 10.1, our study provides clear directions for improvement and quantifies potential benefits. Our approach does not only allow for more ride requests to be accommodated, but also reduces overall travel distance and the number of required vehicles. These outcomes lead to cost savings on different components. Firstly, automating the planning process minimizes the need for manual scheduling work, reducing labour costs. Secondly, fewer vehicles and less fuel are needed. These advantages not only save money but also contribute to Defence's environmental goals by decreasing greenhouse gas emissions.

10.3 Future work

In this section, we explore potential directions for further research and development, building upon the findings and discussions presented in the previous sections. First, we will look at how we can address discussed limitations. Then, we will discuss future work on more enhanced local search methods, and discuss research directions for dynamic environments. Finally, we will consider research possibilities that are relevant to the Ministry of Defence.

Solving limitations To start, many of the limitations discussed in Section 10.1 could trivially be addressed through future work. First, it would be valuable to investigate how the algorithm performs with more realistic and consistent data, which for example eliminates the need for data editing and estimating travel times and distances. Second, enhancing the robustness could be achieved by adding a component to the objective function that quantifies robustness based on slack time, similar to van Twist, van den Akker, and Hoogeveen [70] (2021). However, this would probably not solve the problem with exceeding maximum ride times when delays are added, as discussed in Section 9.3.3. A simple solution to deal with this,

would be decreasing the value α, to decrease the degree of ride-sharing. Further experimentation could show if this would indeed let us better handle delays. Since solutions including ride-sharing are preferred, it might also be worthwhile to investigate the more intricate approach of handling stochasticity in the local search; see for example van den Akker, van Blokland, and Hoogeveen [5] (2013), Passage [52] (2016), and van den Broek, Hoogeveen, and van den Akker [14] (2018). Lastly, the issue of missing a ground truth to compare the quality of our solution, can be solved by applying column generation and local search with recombination through ILP, where the routes obtained through our algorithm function as columns; see for example ten Bosch, Hoogeveen, and van Kooten Niekerk [10] (2021).

Enhanced Local Search As discussed in Section 10.1, operators I2 and I3 have proven effective, but introducing more complex operators could expand the range of options and potentially lead to better solutions. For instance, exploring a more advanced version of I3 that allows the removal of multiple non-consecutive nodes from a route could be worth investigating. Additionally, considering the use of infeasible solutions temporarily in the optimization process could increase the possibilities for finding better solutions. Furthermore, combining simulated annealing with a genetic algorithm, as demonstrated by Masmoudi, Hosny, Braekers, and Dammak [40] (2016) discussed in Section 2, could be a promising avenue for improving the optimization process.

Dynamic Environment Considerations Although we looked at the addition, removal, and delays of rides in existing solutions in Section 8.3, there is room for further exploration in the context of dynamic route adjustments. Future research could examine more sophisticated techniques for modifying established routes. This could involve studying more advanced insertion heuristics and applying smaller-scale local search strategies, similar to what van Twist, van den Akker, and Hoogeveen [70] (2021) and Pouls, Meyer, and Glock [54] (2021) do.

Defence-specific Enhancements With a focus on the Ministry of Defence's specific needs, there are interesting possibilities for expanding this work. First, Defence would be interested in considering additional user preferences, such as a maximum number of ride-sharing arrangements. Second, investigating refuelling constraints would make the model more realistic. Additionally, as the shift toward electric charging takes place, exploring optimal charging station placement across pool locations could be valuable. Simulation studies could help identify the best charging strategies. Moreover, it could be worthwhile to explore the potential of establishing central charging hubs across the Netherlands, independent of pool

locations, for frequently travelled routes.

Furthermore, although we showed in Figure 5.10 in Section 5.2.3 a clear correlation between available vehicles and departures per pool location, the findings in Section 9.1 indicate the potential occurrence of vehicle shortages at certain pool locations, while others might be experiencing a surplus. An interesting area for future research involves examining an improved overall distribution of vehicles among these pool locations. This investigation could be extended by optimizing distributions on a daily basis, using predictive methods such as machine learning to predict for future days the nature of ride requests.

Chapter 11

Conclusion

In this thesis, we solved a unique ride-sharing problem for the Dutch Ministry of Defence. The ministry manages a fleet of pool vehicles distributed strategically across different locations in the Netherlands. Ministry of Defence employees can use these pool vehicles for work-related appointments. With an increasing emphasis on decreasing greenhouse gas emissions and the limitations on vehicle availability, the optimization of vehicle utilization is important. Therefore, we have developed an ILS-algorithm that uses SA and generates schedules that allow for ride-sharing. Additionally, we have introduced the concept of decoupling vehicles from their original pool locations, thereby enabling the option of returning vehicles to different pool locations, and additionally the option of single rides. The ride-sharing problem we have addressed is an extension of the original ride-sharing problem, considering different vehicle types, multiple depots, strict time constraints, user preferences, capacity limitations, and flexible roles of drivers and riders. The multi-objective function is designed to minimize the number of non-planned users, the ratio of total travel distance with and without ride-sharing, the deviation from the desirable vehicle distribution at pool locations, and the number of non-empty vehicles.

An important aspect handled in this thesis is the preprocessing of data. We have shown how we employed a string similarity algorithm, categorized ride types, and established time windows. Additionally, we have discussed forming a distance matrix with estimated ride times and distances derived from a regression analysis in which we used geographic coordinates. Furthermore, we have introduced an initial solution heuristic, an SA-algorithm called `CombinedApproach`, along with explanation of its underlying assumptions, neighbourhood operators, and constraint validation. We have also introduced three comparative algorithms, namely `NaiveApproach`, `RideSharingOnly`, and `UnboundVehicles`. Notably, `NaiveApproach` mimics the current naive approach of planning, while `RideSharingOnly` and `UnboundVehicles` respectively implement exclusively ride-sharing and the decoupling of pool vehicles

from their designated locations.

In the first experiment, we compared the performance of these four algorithms. We found that the `CombinedApproach` outperforms the other algorithms in terms of planned users, with a significantly better distance ratio compared to `RideSharingOnly`. This indicates that decoupling vehicles from pool locations and enabling ride-sharing lead to substantial improvements when combined. Also, when comparing `CombinedApproach` with `RideSharingOnly`, the combined approach has the advantage of being able to plan users who have a bound on a different day. Notably, `CombinedApproach` employs fewer vehicles than the other algorithms in general and subsequently obtains a higher rides-per-vehicle average. These advancements come at the moderate cost of a small average occupation deviation and a slight increase in average ride time. In the second experiment, we showed that `CombinedApproach` effectively handles single rides by scheduling nearly all of them on all days. Furthermore, on many days `CombinedApproach` reduced both the number of non-planned users and the distance ratio compared to the situation where single rides are not included. This is likely attributed to the fact that single rides are suitable for ride-sharing. Lastly, in Experiment 3, we showed that adding rides to `CombinedApproach`-derived solutions in a dynamic context was generally not a problem because of available empty routes. However, we showed removal of rides proved to be more challenging, particularly on busy days with complex ride interdependencies. Nevertheless, a portion of rides could be removed on all observed days, in which the proposed removal heuristic helped. Moreover, even though the introduction of delays during peak hours maintained the feasibility of the majority of routes, a consistent proportion of routes did become infeasible on all days. This is primarily due to exceeding maximum ride time caused by ride-sharing arrangements. To potentially increase robustness, future research could focus on incorporating a slack component in the objective, experimenting with a lower α-value, or applying more intricate approaches.

The ride-sharing problem addressed in this research is an extension of the original problem, and broader in scope than most literature. Besides, literature predominantly deals with smaller instances in terms of requests, depots, and vehicles compared to this study. Moreover, the results could contribute to the development of Ministry of Defence's future planning systems. The proposed algorithm offers the potential to minimize manual scheduling efforts, thereby reducing labour costs. Additionally, this approach leads to a reduction in vehicle utilization and fuel consumption. These outcomes not only translate to cost savings, but also align with the ministry's environmental objectives by decreasing greenhouse gas emissions.

Appendix A

Experiment 1 Tables

Measure	17 Jan	18 Jan	19 Jan	20 Jan	21 Jan	22 Jan	23 Jan
#Users in additional	510	1966	1852	1406	558	31	50
#Users in original	2039	2274	2156	2329	1196	395	200
Next day's vehicle absence	284	283	272	215	126	117	185
#Empty routes	450	386	386	355	674	1113	1078
Average #rides per vehicle	1.921, (0.836)	2.131, (0.756)	2.106, (0.699)	2.058, (0.683)	1.685, (0.606)	1.620, (0.552)	1.244, (0.499)

Table A.1: Experiment 1 results: `NaiveApproach` week 1

Measure	24 Jan	25 Jan	26 Jan	27 Jan	28 Jan
#Users in additional	925	2030	1530	1265	532
#Users in original	1883	2335	2390	2355	1185
Next day's vehicle absence	298	292	269	225	152
#Empty routes	541	358	360	353	711
Average #rides per vehicle	2.158, (0.787)	2.163, (0.704)	2.105, (0.627)	2.046, (0.630)	1.723, (0.622)

Table A.2: Experiment 1 results: `NaiveApproach` week 2

Measure	17 Jan	18 Jan	19 Jan	20 Jan	21 Jan	22 Jan	23 Jan
Fitness initial solution	62402	78932	81340	68525	62025	51327	52030
Fitness ILS round 1	61013	71908	72295	64043	59360	51315	51665
Fitness ILS round 2	60953	71121	71897	63822	59359	51315	51663
Fitness ILS round 3	60949	70900	71721	63758	59359	51315	51663
Fitness ILS round 4	60949	70799	71688	63758	59359	51315	51663
Runtime (min)	206	191	243	195	229	362	425
#Users in additional	529	997	1054	659	469	81	100
#Users in original	2020	3243	2954	3076	1285	345	150
Average absolute deviation	0.941, (0.797)	1.111, (0.839)	0.915, (0.803)	0.954, (0.789)	0.706, (0.706)	0.719, (0.702)	0.654, (0.710)
#Empty routes	741	473	493	519	858	1148	1140
Average #rides per vehicle	3.190, (1.647)	3.272, (1.553)	3.185, (1.647)	3.300, (1.623)	2.697, (1.499)	2.0526, (0.548)	1.862, (1.044)

Table A.3: Experiment 1 results: `UnboundVehicles` week 1

Measure	24 Jan	25 Jan	26 Jan	27 Jan	28 Jan
Fitness initial solution	65001	78431	73828	67964	61325
Fitness ILS round 1	62160	70991	67138	62624	59486
Fitness ILS round 2	62160	70525	67055	62548	59486
Fitness ILS round 3	62160	70510	66962	62404	59474
Fitness ILS round 4	62160	70263	66950	62288	59474
Runtime (min)	161	188	200	200	234
#Users in additional	575	980	830	602	476
#Users in original	2233	3385	3090	3018	1241
Average absolute deviation	1.072, (0.779)	1.046, (0.701)	0.771, (0.693)	0.745, (0.654)	0.732, (0.688)
#Empty routes	685	464	527	586	890
Average #rides per vehicle	3.185, (1.618)	3.387, (1.600)	3.326, (1.678)	3.454, (1.602)	2.905, (1.400)

Table A.4: Experiment 1 results: `UnboundVehicles` week 2

Measure	17 Jan	18 Jan	19 Jan	20 Jan	21 Jan	22 Jan	23 Jan
Fitness initial solution	68616	77111	79029	72806	66423	50625	52951
Fitness ILS round 1	68470	74695	78607	70776	66368	50625	52951
Fitness ILS round 2	68470	74695	78607	70711	66368	50625	52951
Fitness ILS round 3	68470	74695	78607	70711	66368	50625	52951
Fitness ILS round 4	68470	74695	78607	70711	66368	50625	52951
Runtime (min)	141	119	122	174	157	93	103
#Users in additional	947	1375	1559	1114	860	72	190
#Users in original	1602	2865	2449	2621	894	354	60
Ratio ride-sharing	0.999	0.947	0.955	0.973	0.995	1.000	1.000
#Empty routes	778	469	515	531	926	1150	1185
Average #rides per vehicle	2.740, (1.165)	2.887, (1.371)	2.801, (1.205)	2.877, (1.318)	2.444, (0.980)	2.000, (0.000)	2.000, (0.000)
Ratio travel time	1.098, (0.099)	1.100, (0.096)	1.099, (0.098)	1.098, (0.098)	1.098, (0.099)	1.101, (0.101)	1.100, (0.102)

Table A.5: Experiment 1 results: `RideSharingOnly` week 1

Measure	24 Jan	25 Jan	26 Jan	27 Jan	28 Jan
Fitness initial solution	69810	77112	77117	71845	66197
Fitness ILS round 1	69501	74575	75697	70538	66138
Fitness ILS round 2	69501	74575	75697	70457	66138
Fitness ILS round 3	69501	74575	75697	70457	66138
Fitness ILS round 4	69501	74575	75697	70457	66138
Runtime (min)	131	112	117	153	159
#Users in additional	1009	1375	1348	1068	840
#Users in original	1799	2990	2572	2552	877
Ratio ride-sharing	0.994	0.944	0.979	0.987	0.999
#Empty routes	725	452	524	576	942
Average #rides per vehicle	2.756, (1.187)	2.934, (1.412)	2.777, (1.258)	2.881, (1.263)	2.513, (0.992)
Ratio travel time	1.099, (0.098)	1.099, (0.096)	1.098, (0.097)	1.099, (0.098)	1.099, (0.098)

Table A.6: Experiment 1 results: `RideSharingOnly` week 2

Measure	17 Jan	18 Jan	19 Jan	20 Jan	21 Jan	22 Jan	23 Jan
Fitness initial solution	51690	63718	63400	54874	53068	50794	49067
Fitness ILS round 1	43976	48533	45741	43666	45764	47882	30689
Fitness ILS round 2	43458	46948	45140	43064	45452	47882	30689
Fitness ILS round 3	43332	46535	44251	42595	45324	47882	30689
Fitness ILS round 4	43332	46344	43872	42369	44931	47882	30689
Runtime (min)	217	199	211	210	217	190	160
#Users in additional	362	522	480	335	433	66	233
#Users in original	2187	3718	3528	3400	1321	360	17
Ratio ride-sharing	0.725	0.715	0.683	0.712	0.731	0.940	0.531
Average absolute deviation	0.458, (0.628)	0.588, (0.58)	0.523, (0.65)	0.536, (0.607)	0.549, (0.561)	0.601, (0.600)	0.588, (0.591)
#Empty routes	798	522	548	617	924	1148	1201
Average #rides per vehicle	3.936, (2.206)	4.018, (2.22)	4.157, (2.197)	4.252, (2.298)	3.391, (1.778)	2.035, (0.626)	3.000, (1.155)
Ratio travel time	1.102, (0.087)	1.107, (0.088)	1.113, (0.088)	1.114, (0.090)	1.119, (0.091)	1.106, (0.101)	1.092, (0.091)

Table A.7: Experiment 1 results: `CombinedApproach` week 1

Measure	24 Jan	25 Jan	26 Jan	27 Jan	28 Jan
Fitness initial solution	54868	61099	59506	56923	54856
Fitness ILS round 1	46733	48359	48042	46597	47380
Fitness ILS round 2	46514	47811	47516	46468	47267
Fitness ILS round 3	46514	47516	46874	46298	47173
Fitness ILS round 4	46514	47051	46663	46032	47088
Runtime (min)	171	210	257	209	216
#Users in additional	437	565	487	444	474
#Users in original	2371	3800	3433	3176	1243
Ratio ride-sharing	0.758	0.714	0.739	0.745	0.759
Average absolute deviation	0.523, (0.563)	0.418, (0.533)	0.444, (0.524)	0.418, (0.508)	0.471, (0.586)
#Empty routes	748	542	592	650	950
Average #rides per vehicle	3.91, (2.121)	4.299, (2.356)	4.077, (2.183)	4.110, (2.266)	3.471, (1.790)
Ratio travel time	1.099, (0.088)	1.109, (0.089)	1.107, (0.090)	1.112, (0.090)	1.118, (0.091)

Table A.8: Experiment 1 results: `CombinedApproach` week 2

Appendix B

Experiment 2 Tables

Measure	17 Jan	18 Jan	19 Jan	20 Jan	21 Jan	22 Jan	23 Jan
Fitness initial solution	60364	80289	80313	68288	61401	50332	48503
Fitness ILS round 1	58841	71279	71458	62814	58600	50083	48028
Fitness ILS round 2	58841	70854	70964	62218	58574	50083	48028
Fitness ILS round 3	58841	70716	70964	61900	58551	50083	48028
Fitness ILS round 4	58841	70627	70964	61883	58551	50083	48028
Runtime (min)	103	153	141	189	220	176	185
#Users in additional	539	1069	1098	661	496	63	120
#Users in original	2010	3171	2910	3074	1258	363	130
#Single-ride users in additional	13	48	32	12	3	0	1
#Single-ride users in original	87	97	95	85	31	2	8
Average absolute deviation	0.980, (0.892)	1.163, (0.815)	0.941, (0.788)	0.863, (0.770)	0.680, (0.614)	0.614, (0.630)	0.824, (0.727)
#Empty routes	727	468	483	493	868	1145	1138
Average #rides per vehicle	3.199, (1.681)	3.305, (1.619)	3.173, (1.620)	3.246, (1.602)	2.786, (1.489)	2.000, (0.552)	1.657, (0.845)

Table B.1: Experiment 2 results: `UnboundVehicles` week 1

Measure	24 Jan	25 Jan	26 Jan	27 Jan	28 Jan
Fitness initial solution	64333	80201	73803	69322	63548
Fitness ILS round 1	60788	71841	67229	62620	60456
Fitness ILS round 2	60731	71446	67001	62374	60456
Fitness ILS round 3	60731	70938	66959	62303	60456
Fitness ILS round 4	60731	70894	66959	62303	60456
Runtime (min)	147	184	189	203	116
#Users in additional	629	1061	916	650	519
#Users in original	2179	3304	3004	2970	1253
#Single-ride users in additional	21	43	37	17	3
#Single-ride users in original	106	70	113	84	52
Average absolute deviation	1.124, (0.755)	1.046, (0.772)	0.967, (0.773)	1.216, (0.725)	0.824, (0.753)
#Empty routes	690	471	512	580	892
Average #rides per vehicle	3.268, (1.638)	3.431, (1.671)	3.299, (1.640)	3.443, (1.652)	2.946, (1.414)

Table B.2: Experiment 2 results: `UnboundVehicles` week 2

Measure	17 Jan	18 Jan	19 Jan	20 Jan	21 Jan	22 Jan	23 Jan
Fitness initial solution	51510	64108	66013	54278	52918	49673	48065
Fitness ILS round 1	44040	47228	45452	43403	45316	47162	31902
Fitness ILS round 2	43631	46453	44857	42771	44948	47162	31902
Fitness ILS round 3	43326	46008	44459	42346	44819	47162	31902
Fitness ILS round 4	43280	45554	44103	42074	44744	47162	31902
Runtime (min)	232	248	210	210	221	184	188
#Users in additional	374	483	495	352	410	68	232
#Users in original	2275	3902	3640	3480	1378	360	27
#Single-ride users in additional	2	4	4	1	3	0	3
#Single-ride users in original	98	141	123	96	31	2	6
Ratio ride-sharing	0.719	0.714	0.681	0.700	0.737	0.926	0.556
Average absolute deviation	0.471, (0.608)	0.549, (0.638)	0.562, (0.616)	0.523, (0.575)	0.471, (0.551)	0.536, (0.574)	0.562, (0.572)
#Empty routes	786	508	537	596	907	1151	1197
Average #rides per vehicle	3.981, (2.154)	4.129, (2.268)	4.223, (2.331)	4.248, (2.349)	3.342, (1.876)	2.093, (0.652)	2.750, (1.035)
Ratio travel time	1.101, (0.086)	1.105, (0.088)	1.111, (0.088)	1.113, (0.089)	1.117, (0.091)	1.111, (0.1)	1.077, (0.078)

Table B.3: Experiment 2 results: `CombinedApproach` week 1

Measure	24 Jan	25 Jan	26 Jan	27 Jan	28 Jan
Fitness initial solution	54267	59771	61660	55392	54415
Fitness ILS round 1	46666	48171	46661	45717	47745
Fitness ILS round 2	46202	46925	46428	45145	47232
Fitness ILS round 3	46133	46552	46060	45062	47105
Fitness ILS round 4	45970	46212	45689	44605	46930
Runtime (min)	281	220	210	206	212
#Users in additional	446	545	507	400	442
#Users in original	2489	3933	3563	3321	1330
#Single-ride users in additional	3	3	5	7	1
#Single-ride users in original	124	110	145	94	54
Ratio ride-sharing	0.741	0.704	0.711	0.733	0.768
Average absolute deviation	0.667, (0.585)	0.536, (0.500)	0.405, (0.555)	0.444, (0.537)	0.458, (0.562)
#Empty routes	745	510	559	632	926
Average #rides per vehicle	4.065, (2.294)	4.258, (2.397)	4.014, (2.197)	4.138, (2.186)	3.391, (1.808)
Ratio travel time	1.097, (0.088)	1.107, (0.088)	1.108, (0.087)	1.111, (0.09)	1.114, (0.090)

Table B.4: Experiment 2 results: `CombinedApproach` week 2

Appendix C

Experiment 2 Additional Figures

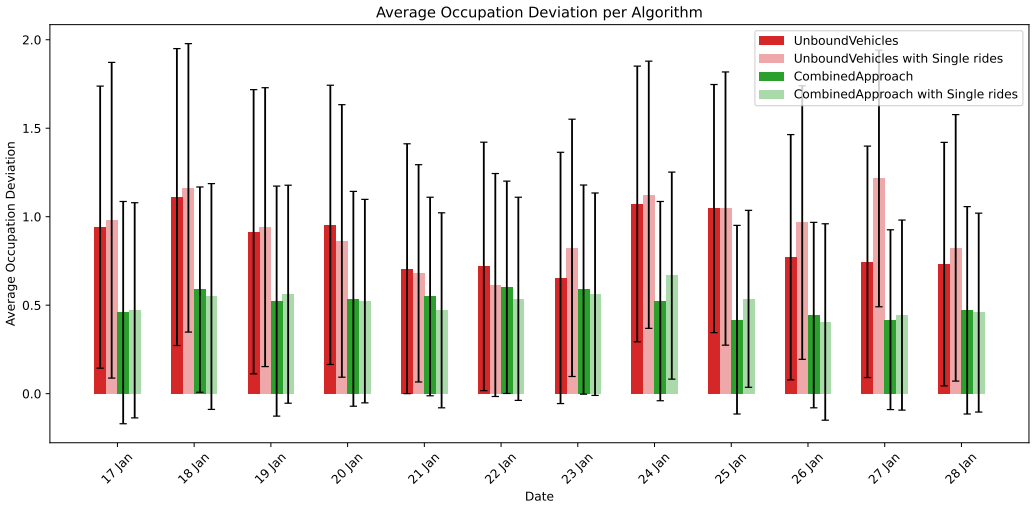

Figure C.1: Difference in average absolute occupation deviation with single rides and without single rides

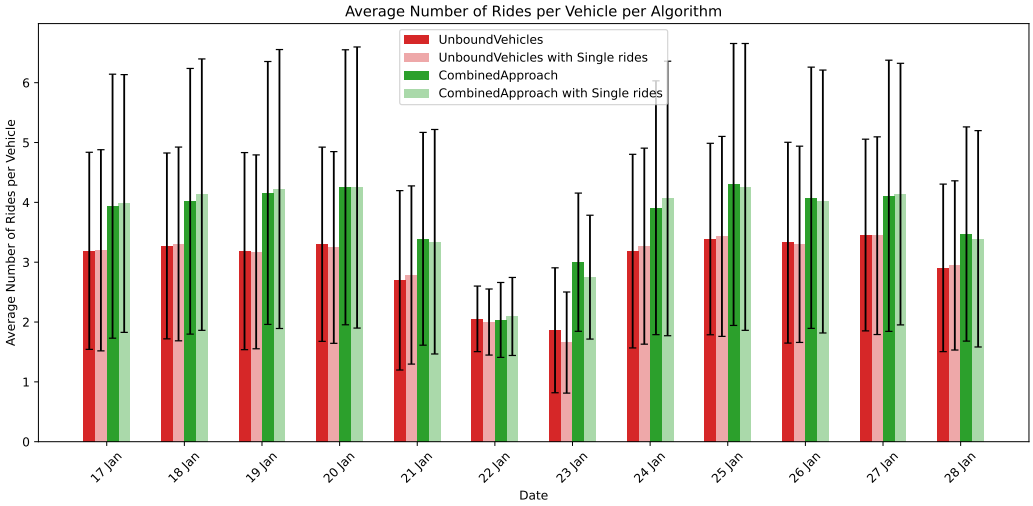

Figure C.2: Difference in average number of rides per vehicle with single rides and without single rides

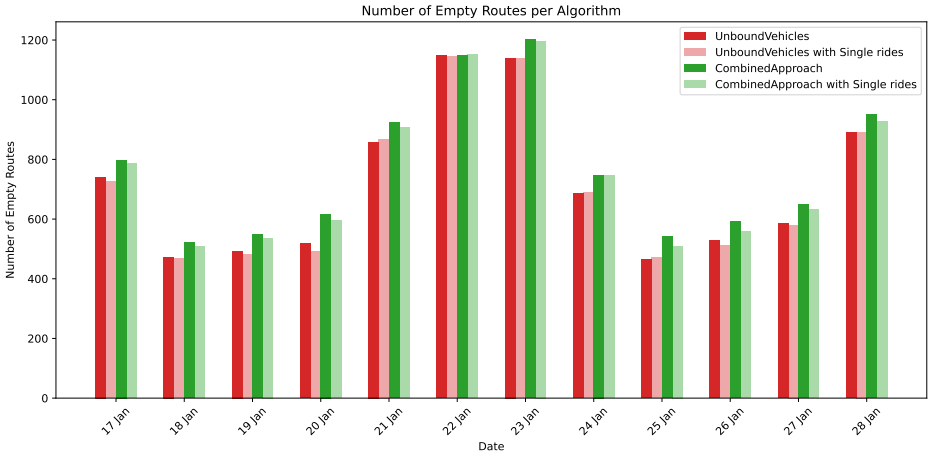

Figure C.3: Difference in number of empty routes with single rides and without single rides

Bibliography

[1] L Abbatecola, MP Fanti, and W Ukovich. "A review of new approaches for dynamic vehicle routing problem". In: *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2016, pp. 361–366.

[2] N Agatz, A Erera, M Savelsbergh, and X Wang. "Optimization for dynamic ride-sharing: A review". In: *European Journal of Operational Research* 223.2 (2012), pp. 295–303.

[3] N Agatz, A Erera, M Savelsbergh, and X Wang. "Sustainable passenger transportation: Dynamic ride-sharing". In: (2010).

[4] N Agatz, AL Erera, MW Savelsbergh, and X Wang. "Dynamic ride-sharing: A simulation study in metro Atlanta". In: *Procedia-Social and Behavioral Sciences* 17 (2011), pp. 532–550.

[5] M van den Akker, K van Blokland, and H Hoogeveen. "Finding robust solutions for the stochastic job shop scheduling problem by including simulation in local search". In: *Experimental Algorithms: 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings 12*. Springer. 2013, pp. 402–413.

[6] M Alarçin and İ Kirçova. "A Conceptual Study On car-sharing Services Based On Sharing Economy". In: *Business & Management Studies: An International Journal* 8.5 (2020), pp. 4521–4545.

[7] V Armant and KN Brown. "Minimizing the driving distance in ride sharing systems". In: *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. IEEE. 2014, pp. 568–575.

[8] R Baldacci, V Maniezzo, and A Mingozzi. "An exact method for the car pooling problem based on lagrangean column generation". In: *Operations research* 52.3 (2004), pp. 422–439.

[9] S Ben Cheikh, C Tahon, and S Hammadi. "An evolutionary approach to solve the dynamic multihop ridematching problem". In: *Simulation* 93.1 (2017), pp. 3–19.

[10] W ten Bosch, JA Hoogeveen, and ME van Kooten Niekerk. “Scheduling electric vehicles by Simulated Annealing with recombination through ILP”. In: *None* (2021). Submitted for publication.

[11] R Botsman and R Rogers. “What’s mine is yours”. In: *The rise of collaborative consumption* 1 (2010).

[12] K Braekers, A Caris, and GK Janssens. “Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots”. In: *Transportation Research Part B: Methodological* 67 (2014), pp. 166–186.

[13] K Braekers and AA Kovacs. “A multi-period dial-a-ride problem with driver consistency”. In: *Transportation Research Part B: Methodological* 94 (2016), pp. 355–377.

[14] R van den Broek, H Hoogeveen, and M van den Akker. “How to measure the robustness of shunting plans”. In: *18th workshop on algorithmic approaches for transportation modelling, optimization, and systems (atmos 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[15] W Chen, M Mes, M Schutten, and J Quint. “A ride-sharing problem with meeting points and return restrictions”. In: *Transportation science* 53.2 (2019), pp. 401–426.

[16] M Cheng. “Sharing economy: A review and agenda for future research”. In: *International Journal of Hospitality Management* 57 (2016), pp. 60–70.

[17] G Clarke and JW Wright. “Scheduling of vehicles from a central depot to a number of delivery points”. In: *Operations research* 12.4 (1964), pp. 568–581.

[18] JF Cordeau. “A branch-and-cut algorithm for the dial-a-ride problem”. In: *Operations Research* 54.3 (2006), pp. 573–586.

[19] JF Cordeau and G Laporte. “A tabu search heuristic for the static multi-vehicle dial-a-ride problem”. In: *Transportation Research Part B: Methodological* 37.6 (2003), pp. 579–594.

[20] JF Cordeau and G Laporte. “The dial-a-ride problem (DARP): Variants, modeling issues and algorithms”. In: *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* 1.2 (2003), pp. 89–101.

[21] CE Cortés, M Matamala, and C Contardo. “The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method”. In: *European Journal of Operational Research* 200.3 (2010), pp. 711–724.

[22] L Coslovich, R Pesenti, and W Ukovich. “A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem”. In: *European Journal of Operational Research* 175.3 (2006), pp. 1605–1615.

[23] GB Dantzig and JH Ramser. “The truck dispatching problem”. In: *Management science* 6.1 (1959), pp. 80–91.

[24] P Detti, F Papalini, and GZM de Lara. “A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare”. In: *Omega* 70 (2017), pp. 1–14.

[25] S Feigon and C Murphy. *Shared mobility and the transformation of public transit*. Project J-11, Task 21. 2016.

[26] S Gökay, A Heuvels, and KH Krempels. “A High-level Category Survey of Dial-a-Ride Problems.” In: *VEHITS*. 2019, pp. 594–600.

[27] T Gschwind and M Drexl. “Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem”. In: *Transportation Science* 53.2 (2019), pp. 480–491.

[28] T Gschwind and S Irnich. “Effective handling of dynamic time windows and its application to solving the dial-a-ride problem”. In: *Transportation Science* 49.2 (2015), pp. 335–354.

[29] J Hamari, M Sjöklint, and A Ukkonen. “The sharing economy: Why people participate in collaborative consumption”. In: *Journal of the association for information science and technology* 67.9 (2016), pp. 2047–2059.

[30] P Healy and R Moll. “A new extension of local search applied to the dial-a-ride problem”. In: *European Journal of Operational Research* 83.1 (1995), pp. 83–104.

[31] W Herbawi and M Weber. “Ant colony vs. genetic multiobjective route planning in dynamic multi-hop ridesharing”. In: *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*. IEEE. 2011, pp. 282–288.

[32] W Herbawi and M Weber. “Evolutionary multiobjective route planning in dynamic multi-hop ridesharing”. In: *European conference on evolutionary computation in combinatorial optimization*. Springer. 2011, pp. 84–95.

[33] W Herbawi and M Weber. “The ridematching problem with time windows in dynamic ridesharing: A model and a genetic algorithm”. In: *2012 IEEE Congress on Evolutionary Computation*. IEEE. 2012, pp. 1–8.

[34] SC Ho, WY Szeto, YH Kuo, JM Leung, M Petering, and TW Tou. “A survey of dial-a-ride problems: Literature review and recent developments”. In: *Transportation Research Part B: Methodological* 111 (2018), pp. 395–421.

[35] M Hossain. “Sharing economy: A comprehensive literature review”. In: *International Journal of Hospitality Management* 87 (2020), p. 102470.

[36] TY Hu and CP Chang. "A revised branch-and-price algorithm for dial-a-ride problems with the consideration of time-dependent travel cost". In: *Journal of Advanced Transportation* 49.6 (2015), pp. 700–723.

[37] JJ Jaw, AR Odoni, HN Psaraftis, and NH Wilson. "A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows". In: *Transportation Research Part B: Methodological* 20.3 (1986), pp. 243–257.

[38] R Ketabi, B Alipour, and A Helmy. "Playing with matches: vehicular mobility through analysis of trip similarity and matching". In: *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2018, pp. 544–547.

[39] S Ma, Y Zheng, and O Wolfson. "T-share: A large-scale dynamic taxi ridesharing service". In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE. 2013, pp. 410–421.

[40] MA Masmoudi, M Hosny, K Braekers, and A Dammak. "Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem". In: *Transportation Research Part E: Logistics and Transportation Review* 96 (2016), pp. 60–80.

[41] N Masoud and R Jayakrishnan. "A decomposition algorithm to solve the multi-hop peer-to-peer ride-matching problem". In: *Transportation Research Part B: Methodological* 99 (2017), pp. 1–29.

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

[43] R Masson, F Lehuédé, and O Péton. "The dial-a-ride problem with transfers". In: *Computers & Operations Research* 41 (2014), pp. 12–23.

[44] H Min. "The multiple vehicle routing problem with simultaneous delivery and pick-up points". In: *Transportation Research Part A: General* 23.5 (1989), pp. 377–386.

[45] Y Molenbruch, K Braekers, and A Caris. "Typology and literature review for dial-a-ride problems". In: *Annals of Operations Research* 259.1 (2017), pp. 295–325.

[46] S Muelas, A LaTorre, and JM Pena. "A distributed VNS algorithm for optimizing dial-a-ride problems in large-scale scenarios". In: *Transportation Research Part C: Emerging Technologies* 54 (2015), pp. 110–130.

[47] S Muelas, A LaTorre, and JM Peña. "A variable neighborhood search algorithm for the optimization of a dial-a-ride problem in a large city". In: *Expert Systems with Applications* 40.14 (2013), pp. 5516–5531.

[48] WP Nanry and JW Barnes. "Solving the pickup and delivery problem with time windows using reactive tabu search". In: *Transportation Research Part B: Methodological* 34.2 (2000), pp. 107–121.

[49] J Paquette, JF Cordeau, and G Laporte. "Quality of service in dial-a-ride operations". In: *Computers & Industrial Engineering* 56.4 (2009), pp. 1721–1734.

[50] SN Parragh, KF Doerner, and RF Hartl. "Variable neighborhood search for the dial-a-ride problem". In: *Computers & Operations Research* 37.6 (2010), pp. 1129–1138.

[51] SN Parragh, KF Doerner, RF Hartl, and X Gandibleux. "A heuristic two-phase solution approach for the multi-objective dial-a-ride problem". In: *Networks: An International Journal* 54.4 (2009), pp. 227–242.

[52] GJPN Passage. "Combining local search and heuristics for solving robust parallel machine scheduling". In: *None* (2016). Retrieved from https://studenttheses.uu.nl/bitstream/handle/20.500.12932/22475/thesis.pdf?sequence=2 - Master thesis, Utrecht University. The Netherlands.

[53] D Pisinger and S Ropke. "A general heuristic for vehicle routing problems". In: *Computers & operations research* 34.8 (2007), pp. 2403–2435.

[54] M Pouls, A Meyer, and K Glock. "Real-Time Dispatching with Local Search Improvement for Dynamic Ride-Sharing". In: *International Conference on Computational Logistics*. Springer. 2021, pp. 299–315.

[55] HN Psaraftis. "A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem". In: *Transportation Science* 14.2 (1980), pp. 130–154.

[56] PWC. *Sharing or pairing? Growth of the sharing economy*. https://www.pwc.com/hu/en/kiadvanyok/assets/pdf/sharing-economy-en.pdf. Retrieved from https://www.pwc.com/hu/en/kiadvanyok/assets/pdf/sharing-economy-en.pdf. (Last accessed: 24 January 2023). 2015.

[57] LB Reinhardt, T Clausen, and D Pisinger. "Synchronized dial-a-ride transportation of disabled passengers at airports". In: *European Journal of Operational Research* 225.1 (2013), pp. 106–117.

[58] C Riley, P van Hentenryck, and E Yuan. "Real-time dispatching of large-scale ride-sharing systems: Integrating optimization, machine learning, and model predictive control". In: *arXiv preprint arXiv:2003.10942* (2020).

[59] BHO Rios, EC Xavier, FK Miyazawa, P Amorim, E Curcio, and MJ Santos. "Recent dynamic vehicle routing problems: A survey". In: *Computers & Industrial Engineering* 160 (2021), p. 107604.

[60] S Ropke and D Pisinger. "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows". In: *Transportation science* 40.4 (2006), pp. 455–472.

[61] M Salehi Sarbijan and J Behnamian. "Emerging Research Fields in Vehicle Routing Problem: A Short Review". In: *Archives of Computational Methods in Engineering* (2022), pp. 1–19.

[62] MW Savelsbergh. "The vehicle routing problem with time windows: Minimizing route duration". In: *ORSA journal on computing* 4.2 (1992), pp. 146–154.

[63] S Shaheen, A Cohen, I Zohdy, et al. *Shared mobility: current practices and guiding principles*. Tech. rep. United States. Federal Highway Administration, 2016.

[64] V Speidel. "EDP-assisted fleet scheduling in tramp and coastal shipping". In: *Proceedings of the 2nd International Ship Operation Automation Symposium, Washington, DC, August 30-September 2, 1976. Proceedings expected to be available about December 1976.* Vol. 5. Proceeding. 1976.

[65] Statista. *Number of sharing economy users in the United States from 2016 to 2021*. https://www.statista.com/statistics/649459/carsharing-fleet-worldwide/. Retrieved from https://www.statista.com/statistics/ 649459/ carsharing-fleet-worldwide/. (Last accessed: 24 January 2023). 2020.

[66] A Tafreshian and N Masoud. "Trip-based graph partitioning in dynamic ridesharing". In: *Transportation Research Part C: Emerging Technologies* 114 (2020), pp. 532–553.

[67] A Tafreshian, N Masoud, and Y Yin. "Frontiers in service science: Ride matching for peer-to-peer ride sharing: A review and future directions". In: *Service Science* 12.2-3 (2020), pp. 44–60.

[68] M Tamannaei and I Irandoost. "Carpooling problem: A new mathematical model, branch-and-bound, and heuristic beam search algorithm". In: *Journal of Intelligent Transportation Systems* 23.3 (2019), pp. 203–215.

[69] KH Ting, LS Lee, S Pickl, and HV Seow. "Shared mobility problems: a systematic review on types, variants, characteristics, and solution approaches". In: *Applied Sciences* 11.17 (2021), p. 7996.

[70] R van Twist, M van den Akker, and JA Hoogeveen. "Synchronizing transportation of people with reduced mobility through airport terminals". In: *Computers & Operations Research* 125 (2021), p. 105103.

[71] NHM Wilson and NJ Colvin. *Computer control of the Rochester dial-a-ride system*. 77. Massachusetts Institute of Technology, Center for Transportation Studies, 1977.

[72] KI Wong, A Han, and C Yuen. "On dynamic demand responsive transport services with degree of dynamism". In: *Transportmetrica A: Transport Science* 10.1 (2014), pp. 55–73.

[73] Y Xu, L Kulik, R Borovica-Gajic, A Aldwyish, and J Qi. "Highly efficient and scalable multi-hop ride-sharing". In: *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*. 2020, pp. 215–226.

[74] I Zidi, K Mesghouni, K Zidi, and K Ghedira. "A multi-objective simulated annealing for the multi-criteria dial a ride problem". In: *Engineering Applications of Artificial Intelligence* 25.6 (2012), pp. 1121–1131.