CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Mokoš David** Personal ID number: **457838**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master’s thesis details

Master’s thesis title in English:

Online Planner for Food Deliveries

Master’s thesis title in Czech:

Online plánovač pro rozvoz zásilek v gastronomii

Guidelines:

1) Study the literature in the field of vehicle routing problems, specifically vehicle routing problem with time windows (VRPTW)
2) Implement a baseline online food delivery planner based on insertion heuristic
3) Based on your research, choose a method for the online food delivery planner
4) Implement the online food delivery planner, and compare its efficiency with the baseline planner

Bibliography / sources:

[1] P. Toth and D. Vigo, Vehicle Routing: Problems, Methods, and Applications, Second Edition. SIAM, 2014.
[2] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte, “Static pickup and delivery problems: a classification scheme and survey,” TOP, vol. 15, no. 1, pp. 1–31, Jul. 2007, doi: 10.1007/s11750-007-0009-0.
[3] J. Jung, R. Jayakrishnan, and J. Young Park, “Dynamic Shared-Taxi Dispatch Algorithm with Hybrid Simulated Annealing,” Computer-Aided Civil and Infrastructure Engineering, vol. 31, Jun. 2015, doi: 10.1111/mice.12157.
[4] A. A. Syed, B. Kaltenhaeuser, I. Gaponova, and K. Bogenberger, “Asynchronous Adaptive Large Neighborhood Search Algorithm for Dynamic Matching Problem in Ride Hailing Services,” in 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Oct. 2019, p. 3006–3012, doi: 10.1109/ITSC.2019.8916943.
[5] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, “On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment,” PNAS, vol. 114, no. 3, pp. 462–467, Jan. 2017, doi: 10.1073/pnas.1611675114.
[6] S. Muelas, A. LaTorre, and J.-M. Peña, “A distributed VNS algorithm for optimizing dial-a-ride problems in large-scale scenarios,” Transportation Research Part C: Emerging Technologies, vol. 54, pp. 110–130, May 2015, doi: 10.1016/j.trc.2015.02.024.
[7] Y. Luo and P. Schonfeld, “A rejected-reinsertion heuristic for the static Dial-A-Ride Problem,” Transportation Research Part B: Methodological, vol. 41, no. 7, pp. 736–755, Aug. 2007, doi: 10.1016/j.trb.2007.02.003.

Name and workplace of master’s thesis supervisor:

Ing. David Fiedler, Artificial Intelligence Center, FEE

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

Date of master’s thesis assignment: **21.02.2021** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until: **19.02.2023**

Ing. David Fiedler
Supervisor’s signature

Head of department’s signature

prof. Mgr. Petr Páta, Ph.D.
Dean’s signature

III. Assignment receipt

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

.	
Date of assignment receipt	Student’s signature

Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Computer Science

Master's thesis

Online Planner for Food Deliveries

Bc. David Mokoš

Supervisor: Ing. David Fiedler

May 20, 2021

Acknowledgements

First and foremost, I would like to thank my thesis supervisor Ing. David Fiedler, for his patient guidance, advice, and encouragement he has provided while I was writing this thesis. His expertise hugely contributed to both the direction and the result of this work.

I take this opportunity to express gratitude to all department faculty members for their help and support throughout my studies. I also thank my partner Ing. Tereza Branyšová for endless encouragement, support, and attention.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 20, 2021 . . . .

Czech Technical University in Prague
Faculty of Electrical Engineering

Citation of this thesis

Mokoš, David. *Online Planner for Food Deliveries.* Master's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, 2021.

Abstract

The delivery services are rapidly gaining in popularity, which was further accelerated by the COVID-19 pandemic. Food delivery, especially, is a difficult problem, where it is necessary to minimize the delivery time to ensure customer satisfaction. Therefore, efficient route planning and order dispatching are essential. This problem of navigating a fleet of vehicles to serve a set of customers is referred to as the Vehicle Routing Problem (VRP), that has been tackled in the literature for several decades. However, real-life instances of VRP generally cannot be optimally solved in a reasonable time and thus, heuristic approaches need to be employed. In this thesis, we adopted a hybrid adaptive large neighborhood search algorithm originally proposed for people transportation and adapted it to the case of food delivery by introducing specific constraints and objectives. The experimental results show that our algorithm produces better solutions compared to several baselines. In addition, the experiments demonstrate the increased intensification and diversification power of our newly added components.

Keywords Food Delivery, Vehicle Routing Problem, Pickup and Delivery Problem, Operations Research

Abstrakt

Využívání rozvozových služeb neustále nabývá na oblibě, čemuž v posledním roce napomohla i celosvětová pandemie COVID-19. Problematické jsou hlavně rozvážky v oblasti gastronomie, kdy je nezbytné minimalizovat dobu rozvozu, aby byla zajištěna spokojenost zákazníků. Za tímto účelem je nutné zaměřit se především na efektivní plánování tras pro kurýry. Tato problematika spadá pod Vehicle Routing Problem (VRP). V reálném světě však pro VRP nelze ve většině případů najít optimální řešení v přijatelném čase, a proto je třeba aplikovat heuristické metody. V této práci byl převzat hybridní adaptivní algoritmus, původně navržený pro přepravu osob, a přizpůsoben pro případ rozvozu jídla zavedením konkrétních omezení a cílů. Experimentální výsledky ukazují, že navržený algoritmus vytváří lepší řešení ve srovnání s jinými algoritmy. Výsledky navíc ukazují zvýšenou sílu intenzifikace a diverzifikace našich nově přidaných komponent.

Klíčová slova Doručování jídla, Problém vyzvednutí a doručení, Problém navigace vozidel, Operační výzkum

Contents

Introduction **1**

1 Literature Review **3**
1.1 Vehicle Routing Problem . . . 3
1.1.1 Capacitated Vehicle Routing Problem (CVRP) . . . 5
1.1.2 Vehicle Routing Problem with Time Windows (VRPTW) 5
1.1.3 Heterogenous Fleet Vehicle Routing Problem (HFVRP) 6
1.1.4 Pickup and Delivery Problem (PDP) . . . 7
1.1.5 Dial-a-Ride Problem (DARP) . . . 7
1.1.6 Evolution and Quality of Information . . . 8
1.2 Solution Methods . . . 9
1.2.1 Exact Methods . . . 10
1.2.2 Heuristic Methods . . . 12
Constructive Heuristics . . . 13
Metaheuristics . . . 13
Hybrid Algorithms . . . 17
Leveraging Machine Learning . . . 17
1.2.3 Methods for Solving Dynamic Problems . . . 18
1.2.4 Available Solvers . . . 19

2 Problem Definition **21**
2.1 Formal Definition . . . 22
2.1.1 Dynamicity . . . 24
2.2 Broader Context within GoDeliver . . . 25
2.2.1 Features . . . 25
2.2.2 Architecture . . . 27

3 Methodology **29**
3.1 The Algorithm for the PDPTW . . . 29
3.1.1 Cost Function . . . 32

3.1.2 Removal and Insertion Operators 33
3.1.3 Local Search Procedure 35
3.1.4 Adaptive Weight Adjustment 36
3.1.5 Initial Solution 37
3.1.6 Diversification Mechanism 37
3.1.7 Parameter Selection 39
3.2 Methods for Evaluation 39
3.2.1 Evaluation Datasets 39
3.2.2 Baseline Algorithms 42
3.2.3 Metrics 43
3.3 Implementation 44
3.3.1 Planning Algorithm 44
3.3.2 Integration into GoDeliver Pipeline 46

4 Results **49**
4.1 Experimental results 49
4.2 Experiments on Adaptive Weight Adjustment 54

Conclusion **57**

Bibliography **59**

A Acronyms **71**

B Contents of an enclosed CD **73**

C Tables **75**

List of Figures

1.1 An intuitive view of the VRP instance and an example solution. . 4
1.2 The taxonomy of VRP and PDP within transportation-related operations research. 4
1.3 Taxonomy of the VRP. 6
1.4 Taxonomy of vehicle routing problems by information evolution and quality. 9

2.1 An example of a single request. 23
2.2 GoDeliver driver application and real-time tracking dashboard . . 26
2.3 Current state of GoDeliver architecture 28

3.1 An example of a single plan traversal for computing the cost function. 33
3.2 A visualisation of the three crossover operators. 38
3.3 A distribution of drop-off locations in the full request dataset of 7 100 requests. 41
3.4 A distribution of the earliest times of drop-off set by the customers throughout the day in the full request dataset of 7 100 requests. . . 42
3.5 Three baseline algorithms for evaluating the performance of our HALNS algorithm. 43
3.6 A simplified entity relationship diagram of the data model used in Go implementation. 45
3.7 Four operator interfaces used in the Go implementation of the HALNS algorithm. 45
3.8 The Abstract Planner class, and three specific planner implementations in GoDeliver Planner. 46
3.9 The new architecture of GoDeliver Planner. 47

4.1 The evolution of the objective value through running time on 50 and 100 request datasets. 51
4.2 Results of metrics M1 - M5 evaluated on our 5 datasets, averaged from 10 runs of each instance in each dataset. 52

4.3 The tradeoff between the total delay and the total distance for each solution produced by the individual algorithms. 53
4.4 Update of the weights of our four insertion operators during the first 3 000 iterations on a single instance. 55
4.5 Update of the weights of our five removal operators during the first 3 000 iterations on a single instance. 55

List of Tables

3.1 The parameters used in our HALNS algorithm 40

3.2 5 evaluation datasets, each containing 10 different instances 42

4.1 Results of metric M1 - Average (and maximum) delay in minutes at drop-off locations, evaluated on our 5 datasets, averaged from 10 runs on each instance. 50

4.2 Results of metric M2 - Average (and maximum) total distance in kilometers travelled by all drivers, evaluated on our 5 datasets, averaged from 10 runs on each instance. 51

4.3 Quality of produced solutions on instances of 50 requests with different combinations of parameters π_1, π_2 and π_3. Values in bold represent the minima. 56

C.1 Percentage of use within 5 minutes of runtime of all insertion operators of HALNS algorithm. 75

C.2 Percentage of use within 5 minutes of runtime of all removal operators of HALNS algorithm. 76

C.3 Percentage of CPU used by each operator of HALNS algorithm within 5 minutes of runtime. 76

Introduction

Consumer habits have been increasingly shifting towards online and this process has been accelerated by the global 2020 COVID-19 outbreak. Online sales grows by 44 % year by year in the US alone [1]. In addition, these changes in customer behavior are likely to have lasting effects [2].

The category of logistics that has recently increased in popularity is food delivery [3]. The advantages of food delivery services are immense, especially during the COVID-19 pandemic, as they facilitate customer access to ready-to-eat meals and enable restaurants and other food providers to keep operating in times of government restrictions [4].

However, the planning of food delivery is a complex problem due to the dynamic setting that is subject to constant change, and only a few global food delivery platforms are able to dispatch orders and plan routes efficiently to lower the delivery cost while meeting the customer expectations. In contrast, local restaurants struggle to provide customers with a good logistics experience because they lack the necessary tools. As a result, they must turn to these global food delivery platforms, which often come with higher cost and lower quality of service.

GoDeliver[1] is a software solution that aims to solve this issue. It provides local businesses with a tool for managing their delivery fleets, which includes automatic order dispatching, and route planning. However, larger businesses with more than one restaurant, such as fast food chains, require more sophisticated planning algorithms that are able to plan near-optimal routes for tens of drivers consisting of hundreds of customers in a reasonable time, given the dynamic environment. As of today, the planning algorithms used by GoDeliver are not ready for such big instances in a dynamic setting.

The problem of finding optimal routes for a fleet of vehicles is a so-called *vehicle routing problem* which has been extensively studied in the literature over the past 60 years. Several promising algorithms have been proposed for

[1] `https://www.godeliver.co/`

a wide range of variants of this problem, including people transportation, and delivery of goods. While most studies focus on basic variants of the vehicle routing problem, this thesis explores the online pickup and delivery problem with time windows with the emphasis on food delivery in which the customer experience is the most important aspect. Therefore, the primary objective is to deliver the food fresh and with the lowest delay possible. Furthermore, because food delivery is a low-margin business, it is also important to keep in mind the cost of delivery to make profit.

The main goal of this thesis is to propose and implement a planning algorithm based on the state-of-the-art research to solve the online pickup and delivery problem with time windows that enables to tackle the real-world use cases of food delivery. The secondary goal is to integrate this algorithm with GoDeliver such that it can be used in production settings. The results of this study will greatly contribute to improving the GoDeliver system by equipping it with faster and stronger planning power, which will be able to serve larger businesses and contribute to the revolution of last-mile logistics.

The experimental results suggest that our algorithm finds overall better solutions on larger instances compared to several baselines. In addition, it seems to find a good tradeoff between minimizing the delays and the cost of delivery. On the other hand, the proposed algorithm needs considerably longer running time.

This thesis is organized as follows: Chapter 1 provides an overview of the vehicle routing problem and presents existing solution methods from the literature. Chapter 2 defines the problem of planning of food delivery and discusses its specifics. It also explains the need for an advanced planning algorithm in the GoDeliver system pipeline and defines the requirements for this algorithm. Chapter 3 presents the methodology and chapter 4 shows the computational experiments of the proposed algorithm on our evaluation dataset.

Chapter **1**

Literature Review

This chapter provides an overview of the research related to the problem of route planning in food delivery. First, we describe the *vehicle routing problem* and related *pickup and delivery problem* along with their classification. Second, we focus on common solution methods from literature. Finally, we reveal some new methods that leverage machine learning and we mention some already available open-source solvers.

1.1 Vehicle Routing Problem

As adopted from Toth and Paolo (2015) [5], the family of *vehicle routing problems* (VRP) can be defined as the following task:

> **Given** a set of *transportation requests* and a *fleet of vehicles*, **determine** a set of *vehicle routes* to perform all (or some) transportation requests with the given vehicle fleet *at minimum cost*; in particular, decide which *vehicle handles which requests and in which sequence* so that all vehicle routes can be *feasibly* executed.

In other words, the problem is concerned with finding the optimal routes to be followed by a fleet of vehicles to serve a set of customers [6]. To better illustrate the problem, figure 1.1 shows an intuitive view of a VRP instance along with its example solution for four vehicles.

The vehicle routing problem is one of the most studied problems in combinatorial optimization due to its relevance in industry [6]. It belongs to the field of operations research applied to transportation problems. Figure 1.2 shows the classification of VRP within operations research and puts in context the family of pickup and delivery problems that will be introduced later, and which is eminently important for this work.

Numerous real-world applications have demonstrated the significance of computer-generated solutions of VRP in terms of global transportation costs

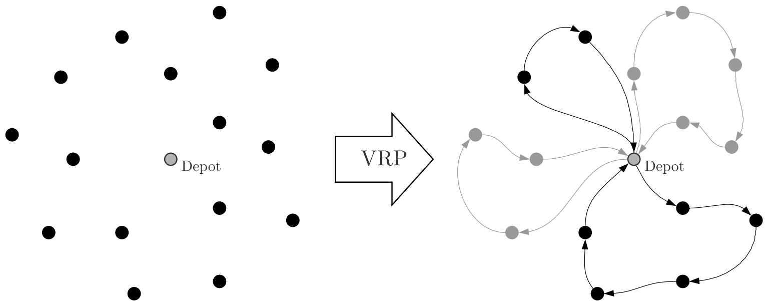

Figure 1.1: An intuitive view of the VRP instance (left) and an example solution for four vehicles (right).

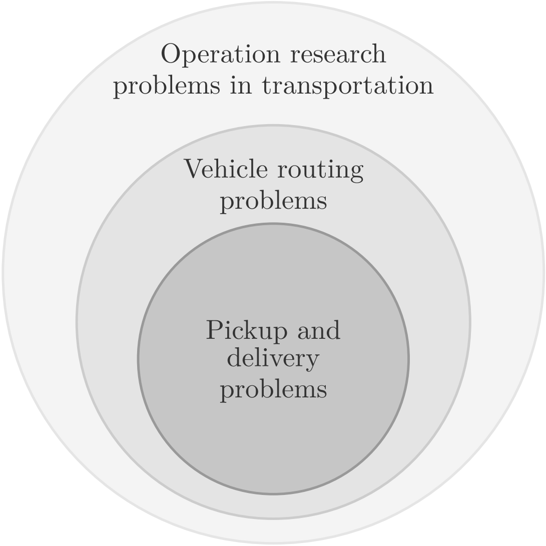
Operation research problems in transportation
Vehicle routing problems
Pickup and delivery problems

Figure 1.2: VRP is one of many problems studied within transportation-related operations research. Pickup and delivery problems are a subclass of vehicle routing problems. The class of operations research contains many other problems apart from VRP, adopted from Ropke (2005) [7, p. 4 (modified)].

[5]. The success of optimization techniques can be attributed not only to the increasing computer power but also to recent research breakthroughs, the development of new mathematical models and to newly adopted approaches that leverage machine learning [5, 8].

The VRP was first introduced in 1959 by Dantzig and Ramser [9] who came with the first mathematical model and an algorithmic approach using a simple matching-based heuristic. Since then, more than 80 years have passed and hundreds of models and algorithms have been proposed for different families of VRP as a result of a large variety of practical applications [5]. This tendency has been enlarged by the burst of consumer delivery in the beginning of the century [8]. To show an example, the American logistics company UPS delivered around 11.5 million packages in 1993, compared to around 6.3 billion in 2020[2] which counts for 24.7 million per single day.

Many variants of VRP have emerged over the years and the literature shows a clear trend towards the study of more complex variants to lower the gap between academic research and real-world applications. These problems imply constraints on various features the route must satisfy, such as vehicle capacity, pickup and drop-off times, or delivery sequence of operations. These classes of routing problems are often called *rich VRPs* [10]. These include *capacitated VRP* (CVRP), in which each vehicle has predefined finite capacity that cannot be exceeded; the *VRP with time windows* (VRPTW), where customers have to be served in predefined time intervals; the *pickup and delivery problem* (PDP), where the goods (or people) to be transported have to be picked up at certain vertices as opposed to the depot; or the *heterogenous fleet VRP* (HFVRP), in which vehicles may have different features, such as different capacities or can deliver only certain type of goods. Routing problems that involve transportation of people are referred to as *dial-a-ride problem* (DARP) (or *dial-a-flight problem* (DAFP) for air transportation) [6,11]. These prevailing variants are detailed in the following sections and their taxonomy is clearly shown in figure 1.3. It is important to mention that these variants make just a fraction of the variants studied in the literature. Moreover, the distinction between the following variants of VRP is not always sharp and many combinations have been derived to more precisely represent real-world problems [5, 6, 12, 13].

1.1.1 Capacitated Vehicle Routing Problem (CVRP)

This variant of routing problems is the most studied in the literature, although it is mostly of academic relevance [5]. In CVRP the requests correspond to the transportation of goods from a single depot to a set of customers. Each customer has a *demand*, which is the amount to be delivered. The vehicle fleet is typically *homogenous*, meaning that all vehicles have the same finite capacity and the same operational cost. Each vehicle starts at the depot, serves a disjunct set of customers, and returns to the depot [5, 15].

1.1.2 Vehicle Routing Problem with Time Windows (VRPTW)

VRPTW is an extension of CVRP where the service at each customer location must occur within predefined time intervals, referred to as *time windows*.

[2] https://stories.ups.com/upsstories/us/en/about-us/global-presence/corporate-facts.html

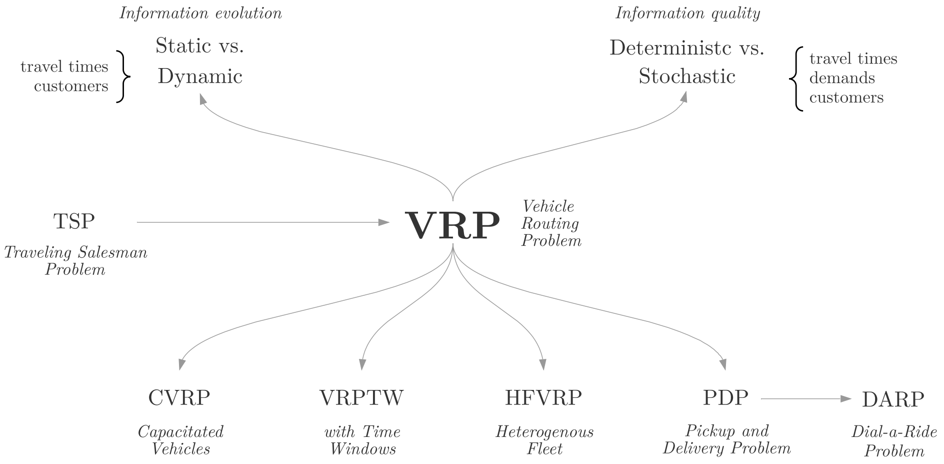

Figure 1.3: Taxonomy of the VRP - the figure shows the prevailing variants of the VRP and common examples of static and dynamic elements, adopted from Bono (2020) [14, p. 26 (modified)].

There are two types of time windows commonly distinguished: soft and hard. Soft time windows can be violated carrying a penalty cost. In case of hard time windows, a driver must not arrive outside of the interval. The driver may arrive to the customer before the time window, however, the customer cannot be serviced until the time window starts. Arriving after the end of the time window is prohibited. Time windows can also be one-sided, i.e., defined as the earliest or the latest time of delivery [5, 12, 16].

1.1.3 Heterogenous Fleet Vehicle Routing Problem (HFVRP)

In this variant, vehicles are characterized by different features, such as different capacities, operational costs, or can only deliver a certain type of goods [17–19]. There are two main branches of HFVRP regarding the obtainable fleet. In the *unlimited* HFVRP, known as *fleet size and mix VRP* (FSMVRP), the problem is to find the best fleet composition and routes, assuming that there is an unlimited number of vehicles of each type. In opposition, in the *limited* HFVRP, known as *heterogeneous VRP* (HVRP), the task is to find the optimal routes for the available fixed vehicle fleet [19]. The cost of the solution usually depends on either the total distance traveled by all vehicles or the total enroute time. This problem is sometimes augmented by introducing time window constraints, making the HFVRP with time windows (HFVRPTW) [18].

In recent years, a relatively new variant related to HFVRP has emerged, called the *green vehicle routing problem* (Green-VRP). The aim of Green-VRP is to minimize fuel consumption and to maximize the use of alternative fuel vehicles, such as electric vehicles, to reduce the impact on the environment. Usually, a heterogenous fleet of both classic and alternative fuel vehicles is considered during the planning process [20].

1.1.4 Pickup and Delivery Problem (PDP)

Three known types of pickup and delivery problems are generally distinguished in the literature [5, 13]. First, the *single-commodity* PDP, where a single type of good is either picked up or dropped-off at each of the customer location. An example of this problem might be transporting money from bank branch offices via an armored vehicle. Second, the *two-commodity* PDP, in which two types of goods are being transported and each customer location may be used for both pickup and drop-off. This problem might occur, for example, in distribution of beverages, where the driver simultaneously delivers full bottles to the customer and collects the empty ones. A variant of the two-commodity PDP is a VRP with backhauls, where a pickup is only allowed when the vehicle is completely empty [16]. Finally, the *n-commodity* PDP is where each type of good is associated with a single pickup location and a single drop-off location. A typical example of this problem can be a food delivery service [13, 21, 22]. In the literature, this problem is often referred to as *vehicle routing problem with pickup and delivery* (VRPPD) [13].

From the implementation perspective, PDP setting introduces a new set of constraints. First, the constraints for the routes, such that the pickup and drop-off must be taken care of by the same vehicle. Second, the precedence constraint, i.e., the pickup happens before the drop-off [14].

A typical extension of PDP emanating from real-life usecases is the PDP with time windows (PDPTW). In this case, each pickup (drop-off) node is associated with a time interval, in which the driver can perform the pickup (drop-off) [16].

1.1.5 Dial-a-Ride Problem (DARP)

The PDP also covers problems associated with people transportation. The major difference between transporting goods and people is the incorporation of customer satisfaction requirements. One of those problems is the so-called *dial-a-ride problem* (DARP). This demand-responsive service is usually provided by a public authority to serve physically impaired people and elderly who may have difficulty using regular transport options [5]. Unlike taxi and public transport services, passengers may share the ride with other passengers and, as a result, may deviate from the direct path from the pickup point to their destination [23]. Some DARPs operate on a heterogenous fleet of vehicles, where different vehicles satisfy the different needs of physically impaired persons, for example, by being wheelchair accessible, some allow transfers from one vehicle to another [23, 24]. The DARP implies that a set of customers make a request, which consists of origin and destination locations and imposes time window constraints for both pickup and drop-off. Additionally, customer inconvenience constraints are typically inflicted, such as maximum enroute time or maximum vehicle capacity [5,24]. There are three objectives of DARP that are often conflicting: maximizing the number of requests the fleet can serve, minimizing the overall operational cost, and minimizing customer inconvenience. A ratio between these objectives is sometimes achieved by first maximizing the number of requests that can be served by the available fleet of vehicles, and then minimizing the operational cost while acknowledging the inconvenience constraints [13].

Other people transportation problems related to DARP include *dial-a-flight Problem* (DAFP) that concerns with air transportation [11], car pooling problem, which consists of finding groups of employees to share a car to work [5], Uber Pool service of an American company Uber [3] that combines taxi service with car pooling [25], or the *school bus routing problem* (SBRP), where children in rural areas are picked up from the bus stop close to their homes and transported to their schools.

1.1.6 Evolution and Quality of Information

In contrast to the classical definition of VRP, real-world use cases often involve two important dimensions. The first one is *evolution*, i.e., whether decisions are made a priori (referred to as *static*) or the information changes in response to new information during the planning process, for instance, when new requests arrive or number of drivers changes (referred to as *dynamic*). Second aspect is the *quality of information*, i.e., whether the information is known with certainty before the start of planning (referred to as *deterministic*) or there exists some amount of uncertainty when decisions are made (referred to as *stochastic*). This categorization applies to VRP as well as DARP. [11, 23]

Based on these dimensions, four categories of routing problems are identified and are shown in figure 1.4. In *static and deterministic* problem, all information is known before the decisions are made and does not change during the execution of planning. It is sometimes referred to as *clasical* problem [11]. *Static and stochastic* problems are the ones where some information is unknown or uncertain at the time of decision, but information about the uncertainty may be known, i.e., in the form of an available range or probability distribution. In addition, routes are planned a priori and only slight changes are allowed during the execution, for example, adding a new node at the end of the plan or skipping a customer. These types of problems do not require any additional technological support. Most of the research in this field

[3] https://www.uber.com/

		Information quality	
		Deterministic input	Stochastic input
Information evolution	Input known in advance	Static and deterministic	Static and stochastic
	Input changes during execution	Dynamic and deterministic	Dynamic and stochastic

Figure 1.4: Taxonomy of vehicle routing problems by information evolution and quality, adopted from Pillac et al. (2013) [11, p. 2 (modified)].

is focused on stochastic customers, stochastic time windows, and stochastic demands [12, 26, 27].

Dynamic and deterministic problems work with some part of the input that is unknown and revealed dynamically during the execution of planning. Thus, the routes for the drivers need to be constantly updated, which requires additional technological support for providing additional information to the planner, such as the location of the drivers using their mobile phones with GPS [12, 28]. Some authors also call these problems *real-time* or *online* [28]. *Dynamic and stochastic* problems are similar to the previous type, but additionally, stochastic knowledge about the dynamically revealed information is available.

1.2 Solution Methods

The approaches to solve vehicle routing problems are generally divided into two categories: exact and heuristic. The exact methods guarantee to find the optimal solution when sufficient resources are provided. However, until now, only small instances of VRP problems involving only up to tens of customers can be solved optimally due to high computational complexity [5, 29]. Indeed, VRP belongs to the class of NP-hard problems [30]. Moreover, according to Savelsbergh (1985) [31], even finding a feasible solution for the VRPTW is itself an NP-complete problem. Nonetheless, some amply constrained problems with reasonable-sized instances can be solved to optimality via mathematical programming techniques [12], which are covered in section 1.2.1.

In real-life problems, exact methods are oftentimes insufficient as large instances need to be solved in a relatively short time. Therefore, heuristic approaches are required to tackle real-world problems. With the trade-off between the computation time and optimality, heuristic algorithms are able to find solutions of relatively good quality within acceptable computational time even for large instances [29]. In addition, heuristic algorithms offer a decent amount of flexibility which allows to target different problem settings that vary in constraints and objectives [5]. However, heuristic algorithms lack the theoretical guarantee regarding the quality of produced solution. Observations about solution quality can only be made empirically based on experiments [7,29].

A class of heuristic algorithms that has gained attraction over the last years is *metaheuristics.* They provide a general framework for heuristics applicable to many problem variants and they often generate solutions of very high quality [7]. In fact, metaheuristics are capable of yielding solutions within seconds whose values lie within one percent from the best known values [5]. Heuristic algorithms are further described in section 1.2.2.

Another special class of heuristics that should be mentioned are *approximation algorithms.* These methods provide both a solution and an error guarantee. For instance, they could guarantee that the obtained solution is at most k times worse than the best obtainable solution [32].

The following sections provide an overview of both exact and heuristic methods for solving the VRP. More emphasis is being put on metaheuristic approaches as they are currently the state-of-the-art solution for most real-world VRP problems [5].

1.2.1 Exact Methods

For many real-world problems, the solution space is so large that even if we had a computer that could evaluate the cost of a trillion solutions per second, and we had started it right after the big bang, around 14 billion years ago, it would not have evaluated all feasible solutions by today. Therefore, exact algorithms need to use other methods rather than simple enumeration [7]. This section lists a few most popular exact methods. For a detailed description and an extensive survey, we refer the readers to Toth and Paolo (2015) [5] and Ho at al. (2018) [23].

The CVRP is the oldest and the most basic variant of the VRP, hence the first exact methods were developed mostly for the CVRP and later extended to be used with other VRP variants. The CVRP is an extension of the known *travelling salesman problem* (TSP), where the task is to find a Hamiltonian circuit with minimum cost while visiting each node exactly once [5]. The TSP was first formulated in 1930, around 30 years earlier than the VRP [9,33]. Therefore, the foundation of many exact algorithms for the CVRP builds on the thorough work done for the exact solution of the TSP [5]. However, as it turns out, the CVRP is significantly harder to solve in practice compared to the TSP. The best exact methods for the CVRP are rarely able to solve instances with over a hundred customers, whereas the TSP instances with thousands of nodes are now routinely solved to optimality [12].

Exact methods directly address the problem using integer linear programming. The objective is to find a binary assignment for decision variables, modelling connections between vehicles and nodes, while minimizing the objective value [14]. The early methods to solve the CVRP were mainly direct tree search algorithms based on branch-and-bound (B&B) [5]. Algorithm 1 shows a main loop of branch-and-bound.

The computational complexity of the B&B can be exponential in the worst case [23], thus some relaxations need to be employed to speedup the search. The combinatorial relaxations used in the branch-and-bound algorithms were adopted from what was earlier proposed for the TSP [34] and later extended by better relaxation techniques based on Lagrangian relaxations and the additive approach. These methods were able to solve instances with lower tens of customers to optimality [5]. Later, branch-and-cut algorithms exploited cutting planes to tighten the linear programming relaxations, which led to a higher chance of finding integer solutions, and provided stronger bounds for optimality verification. In contrast, branch-and-price algorithms focused on column generation rather than generating cuts, which allowed to solve larger mixed-integer programs while still guaranteeing the convergence to global optimum [5, 23]. Current best algorithms for the CVRP belong to the branch-and-price-and-cut family, which combines cuts and column generation, and are much more effective than each of those techniques alone [35].

```
currentBest, upperBound $\leftarrow$ FindInitialSolution(problem);
add problem to queue;
while queue is not empty do
    problem $\leftarrow$ pop item from queue;
    for subProblem in branch do
        if CanSolve(subProblem) then
            candidate, cost $\leftarrow$ Solve(subProblem);
            if cost $\leq$ upperBound then
                currentBest, upperBound $\leftarrow$ candidate, cost;
            end
        end
        else if LowerBound(subProblem) $\leq$ upperBound then
            add subProblem to queue;
        end
    end
end
```

Algorithm 1: Branch-and-Bound, adopted from Bono (2020) [14, p. 34 (modified)].

Branch-and-price is largely used for solving a variety of routing problems and is not constrained to only the CVRP. For the VRPTW, this approach was first used by Desrochers, Desrosiers, and Solomon (1992) [36]. The branch-and-price-and-cut algorithm was also later applied to VRPTW by Kohl et al. (1999) [37].

Similarly, for the pickup and delivery problems, Lu and Dessouky (2004) [38] and later Ropke et al. (2007) [39] have used the branch-and-cut algorithms for the PDPTW. Their approach was able to solve instances of up to 75 customers to optimality. The branch-and-cut-and-price algorithm by Ropke et al. (2009) [40] was able to solve some highly constrained instances of PDPTW of up to 500 customers. In the DARP setting, most of the exact methods are also built upon the concept of branch-and-bound [23]. For multiple vehicles, methods based on branch-and-price seem to be the most promising in the current state of the art. Still, most PDP instances remain much harder to solve than the same-size instances of classical VRP. This is predominantly caused by the precedence constraints, which generally lead to poor linear programming relaxations [5].

The main advantage of using the exact methods is that the optimality of the solution is guaranteed. This is particularly important for static problems. Therefore, the majority of the research in exact algorithms focuses on static and deterministic problems. Computational times of several hours are acceptable for problems, where the information is provided in advance and no changes are expected. Even if the optimal solution cannot be retrieved by the exact algorithm, the planner is able to measure the optimality gap, aka how far the solution can be from the optimum [23].

1.2.2 Heuristic Methods

When problems are not manageable by exact methods, for instance, due to unacceptable computational time or memory issues, heuristics are essential to provide high-quality solutions [23]. The field of VRP heuristics is so rich nowadays that it would be an unimaginable task to describe or even list all of them in this section. Instead, we summarize the most prevailing methods that are still being actively used and present some new and promising techniques.

The progression of heuristic approaches for VRP in the last decade has been primarily in the context of metaheuristics. What best describes this progression is the hybridization of both concepts and scope. First, there has been an emergence of new heuristics that combine several concepts in terms of search principles that have been initially developed individually, such as genetic algorithms, tabu search, large neighborhood search, or simulated annealing. In addition, other strategies and techniques have found their place in the most popular methods. These include exotic large neighborhoods, exact mathematical techniques, decomposition, and cooperation schemes, to name a few. Second, the hybridization of scope in heuristics means that they offer a decent amount of flexibility which allows to target different problem variants with various constraints and objectives without any extensive structural changes [5].

The following two sections summarize some relevant results regarding constructive heuristics and metaheuristics. We also take a brief look at the latest advancements in machine learning methods for VRP.

Constructive Heuristics

Constructive heuristics are frequently utilized to quickly create an initial feasible solution. Although metaheuristics are more effective, constructive heuristics are employed in various dynamic problems that require a feasible solution within milliseconds [41, 42], or to evaluate various operational policies [42]. In many cases, solutions obtained via constructive heuristics are used to provide an initial solution for more complex methods [43, 44]. However, many metaheuristics are so robust that they can be initialized with any random solution, which does not even need to be feasible, and still produce high-quality results [5].

Most constructive heuristics are inspired from the greedy insertion heuristic introduced by Jaw et al. (1986) [45], who was the first to adapt the traditional insertion algorithm to DARP with time windows and multiple vehicles. The algorithm selects the requests sorted by the earliest pickup time and inserts the request into the cheapest feasible position in the existing routes. Alternatively, it adds a new vehicle if no feasible position is found.

Lu and Dessouky (2006) [46] proposed an insertion-based heuristic for the PDPTW, which regarded the deviations from time windows and an increase in travel time. The results show that the heuristic outperforms several decent methods on standard benchmarking problems. A similar approach was taken by Fielder et al. (2018) [47] in a dial-a-ride setting.

Metaheuristics

Metaheuristics for the VRP can be loosely classified into two categories: local search methods and population-based methods. However, the frontiers between these methods have become increasingly fuzzy in recent years and many hybrid methods have emerged. They will also be discussed in this section.

Local search algorithms start from an initial solution x_0 and at each iteration t, they move from the current solution x_t to another solution x_{t+1} in its neighborhood $N(x_t)$. The neighbors are retrieved by applying operators, which make minor modifications of the original solution. If any neighbor yields a better performance, i.e., the cost $c(x_{t+1})$ is lower than $c(x_t)$, it becomes the new candidate solution. However, local search algorithms usually allow worsening of the solutions, such that the cost $c(x_{t+1})$ does not always have to be lower than $c(x_t)$, to better explore the solution space and escape the local optimum. As a result, certain mechanisms must be put in place to avoid cycling [5, 14]. Algorithm 2 shows a common main loop of local search algorithms.

```
candidate ← FindInitialSolution();
best ← candidate;
while stopping criterion is not met do
    if IsValid(candidate) and Cost(candidate) < Cost(best) then
        best ← candidate;
    end
    neighbor ← FindBestNeighbor(candidate);
    if Cost(neighbor) < Cost(candidate) or AllowWorse(neighbor) then
        candidate ← neighbor;
    end
end
```

Algorithm 2: Local Search, adopted from Bono (2020) [14, p. 36 (modified)].

Tabu Search One of the local search algorithms is *tabu search*. It moves from a solution x_t to the best non-tabu solution x_{t+1} in its neighborhood $N(x_t)$. To avoid cycling, it keeps a list of previous candidate solutions that are declared *tabu*, or forbidden, to the completion of the algorithm, or for a certain number of iterations [5].

Tens of different implementations of tabu search have been proposed over the years. One of the first metaheuristics for PDPTW was an algorithm proposed by Nanry and Barnes (2000) [48]. They developed the tabu search with three operators: moving a pickup-drop pair into another plan, switching two pickup-drop pairs between plans and moving individual nodes within the plan. Cordeau and Laporte (2003) [49] were among the first to use the tabu search for the DARP with satisfactory results. Moreover, Zachariadis and Kiranoudis (2010) [50] showed that tabu search can perform very well even on large-scale instances.

Simulated Annealing *Simulated annealing*, another well-known local search operator uses a technique inspired by the physical annealing process. The solution x_{t+1} is usually randomly selected from the neighborhood $N(x_t)$. To avoid being stuck in a local optimum, worse or nonimproving solutions are accepted with a given probability [14]. One known implementation of this algorithm for the VRP is that of Osman (1993) [51].

One of the recent studies conducted by Braekers et al. (2014) [44] proposed a highly effective deterministic variant of simulated annealing, named *deterministic annealing.* In this study, nonimproving solutions are accepted while the deterioration of the objective value is lower than a deterministic threshold. Braekers et al. used more complex operators and, in addition, a restart strategy to escape unattractive regions of the search space.

Variable Neighborhood Search Both the tabu search and simulated annealing work well as standalone algorithms, but are widely used in combination with local search heuristics, such as *variable neighborhood search* [52,53]. This algorithm was first proposed by Mladenović and Hansen (1997) [54]. It works with several neighborhoods $(N_1, \ldots, N_p)$ that are systematically changed in the descent and preturbation phases of the local search. The algorithm starts from an initial solution x_0 and iteratively applies these neighborhoods until no improvement is achievable. After the last applied neighborhood, the cycle restarts. The algorithm usually finishes after a defined number of cycles or, as introduced by Li and Lim (2003) [55], when the solution is no longer improving.

Kytöjoki et al. (2007) [56] were among the first who successfully applied variable neighborhood search to solve the VRP. Two years later, Parragh et al. (2009) [57] proposed a variable neighborhood search for the DARP. Thanks to the great results, more researchers have adapted the algorithm to other DARP usecases in recent years [58–62]. Variable neighborhood search proved to be able to obtain high-quality solutions even for large-scale VRP instances of up to 20,000 customers [5].

(Adaptive) Large Neighborhood Search One of the most successful metaheuristics for a wide variety of routing problems is *large neighborhood search* [63], first introduced by Shaw (1997) [64]. At each iteration, part of the current solution is destroyed (e.g., k requests are removed) by applying one or more removal operators and rebuilt again into a new complete solution using one or more insertion operators [23]. The modifications of solutions are commonly bigger than those of previously described heuristics. In an adaptive extension of this algorithm, called *adaptive large neighborhood search*, the removal and insertion operators are selected based on their past performance during the search, usually by employing a roulette wheel selection process with an adaptive weight adjusting mechanism [63, 65, 66].

Ropke and Pisinger (2006) [67] used the adaptive large neighborhood search heuristic to solve the PDP with time windows. Their algorithm used removal and insertion operators already existing in the literature, including the removal operator by Shaw (1997) [64], and an acceptance criterion for new solutions known from simulated annealing. The algorithm by Ropke and Pisinger has proven to be powerful and has served as a building block for many further studies in complicated routing problems, particularly PDP and DARP [43, 63, 66, 68–74]. Gschwind and Drexl (2016) [63] adopted the algorithm from Ropke and Pisinger and added three more removal operators. They also demonstrated how to test a new solution for feasibility in an amortized constant time. Their version of the adaptive large neighborhood search produced better solutions on standard DARP instances compared to the vast majority of other algorithms, except for the hybrid genetic algorithm by Masmoudi et al. (2017) [75].

The other well-known category of metaheuristics are population-based methods, which are inspired by natural processes, such as the evolution of species or the behavior of insects. All successful population-based heuristics rely on local search methods to drive the search towards promising areas and to avoid local optima. As a result, the majority of population-based algorithms are naturally hybrid [5].

Ant Colony Optimization This metaheuristic has also proven practical for many routing problems. It is inspired by the pheromone mechanism used by ants for coordination. Each ant simulates a solution by traversing the graph along its edges and accumulates pheromons along the edges it traversed. In each iteration, the ants select the edges with a probability proportional to the pheromon value. The pheromon evaporates after a given number of iterations, so the most promising edges remain at the end [14, 76]. The algorithm by Reimann et al. (2004) [77] was one of the most successful.

Ant colony optimization algorithm is particularly interesting in dynamic and stochastic settings. When new information is received, such as a new request arrives, or some delays occur, the algorithm uses the pheromone trails from previous iterations. This relies on the assumption that the new information does not disrupt the current solution too much and that some patterns can still be exploited [14, 78].

Genetic Algorithms *Genetic algorithms* are population-based methods that are inspired by the evolution of species. These algorithms usually work with not a single solution, but rather a population of solutions, called *individuals*. At each iteration, the individuals with better fitness are selected with higher probability to be parents. New individuals are created by applying the crossover and mutation operators to the parents and added to the population, while some other individuals are replaced [14, 23].

The first successful application of genetic algorithms to solve the VRP is that of Prins (2004) [79]. Prins combined the genetic operators, selection and crossover with a local search method to replace the classic random mutation operator. Other researchers successfully applied genetic algorithms for numerous VRP variants, for example, Masmoudi et al. (2017) [75] proposed a hybrid genetic algorithm to tackle the DARP with excellent results, even outperforming in terms of quality of solution, many large neighborhood search algorithms that are usually better performing on larger instances [23].

Hybrid Algorithms

Combining metaheuristics with other metaheuristics or exact methods is a growing trend that has proven successful. Indeed, many state-of-the-art algorithms for combinatorial optimization problems are hybrid algorithms.

In the literature, hybrids of metaheuristics are the most prevalent [23]. They usually come in two forms. First, each metaheuristic is executed sequentially, for example, in a study by Parragh et al. (2009) [57], where several algorithms are applied to the best solutions obtained from the previous algorithms. The other option is that one metaheuristic is executed in another metaheuristic. A popular approach is to embed local search methods into population-based heuristics, which takes advantage of combining the exploration ability of population-based heuristics with the exploitation ability of local search methods. Examples of this approach are the studies by Masmoudi et al. (2016 and 2017) [43, 75].

Hybrids of metaheuristics with exact methods are also employed. They are obtained by either embedding a metaheuristic to the mathematical or constraint programming approach or vice versa. An example could be a research conducted by Parragh and Schmid (2013) [80]. They combined a column generation approach with a variable neighborhood search. To improve the obtained solutions, a large neighborhood search was employed. Another example could be the study by Gschwind and Drexl (2016) [63] in which dynamic programming is used within a large neighborhood search.

Leveraging Machine Learning

Designing a good heuristic algorithm is difficult as it requires expert knowledge of the problem and often involves the incorporation of custom features or constraints. Several alternative approaches have been proposed that use neural networks to learn heuristics directly from data. The performance of these heuristics has been improving in recent years, however, pure machine learning approaches are still usually outperformed by classical optimization methods [29, 81].

One of the early research in this field was by Potvin et al. (1996) [82], who used a competitive neural network model and a genetic algorithm to improve a parallel insertion heuristic introduced by Potvin and Rousseau (1993) [83] for the VRPTW.

Since then, machine learning algorithms have been significantly improved and seem to be the focus of many researchers. Recently, Kool et al. (2019) [84] has proposed a new attention model for learning heuristics for optimization problems and presented a learning mechanism based on reinforce. This model has been one of the building blocks of a research conducted by Peng et al. (2020) [29] who presented a dynamic attention model with dynamic encoder-decoder architecture, which, based on the results, outperforms the attention model presented by Kool et al.

Still, the prevailing problem with machine learning approaches is that they lack flexibility in terms of different features, constraints, and objectives. As in response to this problem, in the last two years, several studies have emerged that have made a notable leap in this area [8, 81, 85, 86].

The capacity constraints were introduced in a recent study by Nazari et al. (2018) [85], who used reinforcement learning to solve the CVRP. Their approach outperforms Google's OR-Tools[4] on medium-sized instances within comparable computational time and has the potential to be used on other VRP variants or even other combinatorial optimization problems. Furthermore, Hottung and Tierney [81] proposed a large neighborhood search combined with a deep neural network with an attention mechanism to solve CVRP. They were able to outperform large neighborhood search methods based on classical optimization techniques on instances of around 300 customers. Recently, time windows were successfully tackled by Falkner and Schmidt-Thieme (2020) [8].

Some other features have been successfully incorporated into machine learning models. Falkner and Schmidt-Thieme (2020) [8] presented an algorithm that successfully solves VRP with time windows. Similarly, Li et al. (2021) [86] showed how to solve PDP via deep reinforcement learning.

All mentioned works use generated instances in a two-dimensional space where the distances between nodes are measured by Euclidian distance. To the best of our knowledge, no study has used real distances and durations between the nodes, as well as more sophisticated features like different service times at each location, different depots, or already picked up deliveries in a dynamic setting. Therefore, more research has to be done to successfully deploy those techniques into practice.

1.2.3 Methods for Solving Dynamic Problems

This section briefly focuses on the recent research of dynamic problems defined in section 1.1.6. In this context, heuristic approaches are prevalent for two reasons. First, the majority of real-world dynamic problems are too complex for exact algorithms to provide a solution in a reasonable time [14]. Second, information is revealed over time and the complete instance is only known at the end of the planning horizon. As a consequence, even though the exact methods can provide an optimal solution for a current instance, they cannot guarantee that the solution remains optimal after new information is revealed [11].

[4] See section 1.2.4 for details about OR-Tools.

Approaches for dynamic VRPs are based on either *periodic reoptimization* or *continuous reoptimization*. Periodic reoptimization approaches start with the initial set of routes and trigger the replanning procedure in response to information changes or at given time intervals. In each replanning procedure, the optimization algorithm works on the static problem. The main disadvantage of this method is that the optimization must be executed on each change, which is time-consuming, and sometimes impractical, given the nature of dynamic problems [11, 23].

Approaches based on continuous reoptimization maintain information about good solutions in an adaptive memory. When the known information is updated, a decision procedure exploits the gathered memory using lookup strategies, such as Monte-Carlo trajectory sampling [14], to produce a new solution. These approaches usually require more complex implementation, however they maximize the computational capacity and have a net advantage compared to the previous category [11, 14].

1.2.4 Available Solvers

There is a number of open-source and commercial solvers available for the most prevailing variants of the VRP. Important questions that arise regarding solving real-world problems are regarding the quality of the produced solution, generality, flexibility, or speed. This section lists the most popular solvers and puts these questions into context.

A popular tool mainly in the industry is *OR-Tools*[5], an open-source project by Google. It can be used to solve various combinatorial optimization problems and it provides wrappers for several languages, including Python, Java, C++, and C#. In the context of VRP, OR-Tools supports many features, such as pickup and delivery, time windows, multiple depots, different start and end locations of drivers, capacity, skills, etc. In addition, Google provides developers with a rich documentation and an active community [87, 88].

There are 14 search strategies that can be selected for the VRP, including simulated annealing, tabu search, or greedy descent. Additionally, 13 different strategies to find an initial solution are available, such as *parallel cheapest insertion*. The *automatic* initial solution strategy lets the solver select the strategy automatically based on the problem variant [87].

The limitations of OR-Tools arise when larger instances need to be tackled. In some cases, changing the search options for the solver or a strategy for finding an initial solution can help, but generally the instances of rich VRPs, such as the PDPTW, are intractable for OR-Tools [88].

Another commonly used open-source software for solving the VRP is *Vehicle Routing Open-source Optimization Machine* (VROOM)[6]. It can find good

[5] `https://developers.google.com/optimization`

[6] `https://github.com/VROOM-Project/vroom`

solutions in small computing times and can scale for larger instances. Besides the common features like time windows and capacity, it supports some infrequent features, such as driver breaks and working hours. It is not only a VRP solver, but it bundles out-of-the-box integration with routing engines to produce cost matrices based on map data, which makes it easier to use in a real-world setting [89].

VROOM includes several heuristics for finding an initial solution that are selected automatically based on the specific use case. These include *Christofides heuristics*, *clustering heuristics*, or *Solomon insertion heuristics*. The optimization algorithm behind VROOM is a local search procedure that uses 14 different exchange and reallocate operators [88, 90].

Yet another open-source software that deserves attention is *jsprit*[7], which is a Java based, open-source toolkit for solving TSP and VRP. Similarly to OR-Tools and VROOM, jsprit can solve problems with all classic features. Additionally, it allows to define additional constraints. It is well-documented and benchmarked on classic VRP instances [91]. The optimization mechanism behind jsprit is based on the large neighborhood search [88].

[7] `https://github.com/graphhopper/jsprit`

Chapter 2

Problem Definition

The literature review introduced the vehicle routing problem and presented the available solution methods. In this chapter, we demonstrate a real-world problem of route planning in food delivery and elaborate the constraints and objectives in detail.

The objective of this work is to come up with a suitable approach for the planning of food delivery. Food delivery may involve several categories, such as recurrent meal box delivery, grocery delivery, or delivery of ingredients. In this article, the term food delivery will be used to refer to online restaurant delivery, which is a courier service in which a restaurant or a third party logistics company delivers food to a customer. This food is typically fresh (hot or cold) and is intended to be eaten right away. Therefore, the food is picked up by the driver soon after it is prepared by the restaurant. The delivery orders are typically on demand, i.e., the customers expect their food to arrive in a short period of time after placing the order. These orders are placed via restaurant websites, a phone call, or a mobile application [92].

The prepared food is prone to damage if dropped, tilted, or kept for a longer duration. Therefore, technology must be involved in the process of food delivery. Food portions are usually packed in plastic or paper containers which are stored in thermal bags or boxes while being carried by the courier. In addition, to reduce the duration from cooking to delivery and, consequently, to ensure customer satisfaction, software tools and planning need to be engaged [92].

The costs associated with the delivery service are paid either by the customer, the restaurant, or are split between both parties. These costs include drivers' salaries, gas costs, vehicle amortization, or other transportation costs. The aim of the planning procedure is to minimize those costs and consequently, increase the revenue of the service. However, in the food delivery business, customer satisfaction is commonly the most important objective, because it results in customer retention, ultimately leading to a scalable and successful business.

From the VRP perspective, food delivery has its own specifics compared to other categories of logistics. The four most important ones with respect to route planning are:

Dynamicity New information is revealed during the day, such as the arrival of new orders, unexpected delays due to traffic, or changing the number of drivers. Therefore, high emphasis is placed on the speed of the selected algorithm, as the drivers need to react to changes as quickly as possible.

Short Time Windows Customers expect their food to arrive shortly after they order it. In addition, ready-to-eat food has short lastingness and needs to be delivered as soon as possible to secure its quality.

Soft Time Window Constraints The time windows for food delivery are generally soft, meaning they can be violated carrying a penalty cost. In addition, visiting a customer before the specified time window is usually preferred over being late, so the penalty is lower in that case.

Peak Times Food delivery has peak times around lunchtime and dinnertime. With this, the number of required drivers need to change dynamically according to the time of the day.

2.1 Formal Definition

It is clear that in the context of food delivery, each customer request consists of transporting goods (food) from one pickup location to one drop-off location. As there may be more restaurants preparing food, rather than a single depot, it is the case of *pickup and delivery problem* (PDP). Since food delivery commonly involves multiple drivers and time window constraints, the ultimate generalization of this problem can be referred to as the *pickup and delivery problem with time windows* (PDPTW). This variant also involves capacity constraints, i.e., how many food portions can each driver carry at any moment. In addition, one feature that is required for our use case is that the drivers can start at arbitrary locations rather than a single depot. An example of a single request is shown in figure 2.1.

Using a similar notation to Cordeau (2006) [93], let us define this problem formally:

We are given a complete graph $G = (V, A)$, where V is the set of nodes and $A = \{(i, j) : i, j \in V, i \neq j\}$ is the set of arcs. V is further partitioned into three subsets $V = S \cup P \cup D$. The set S, $|S| = k$ is the set of start/depot nodes, i.e., the nodes where the couriers/drivers $K = 1, \cdots, k$ are initially located. The positive integer k is the number of couriers/drivers for which the routes are computed. The set P is the set of pickup nodes $P = \{1, \cdots, p\}$, where p is the number of pickup nodes, and D the set of drop-off nodes $D = \{1, \cdots, d\}$, where d is the number of drop-off nodes.

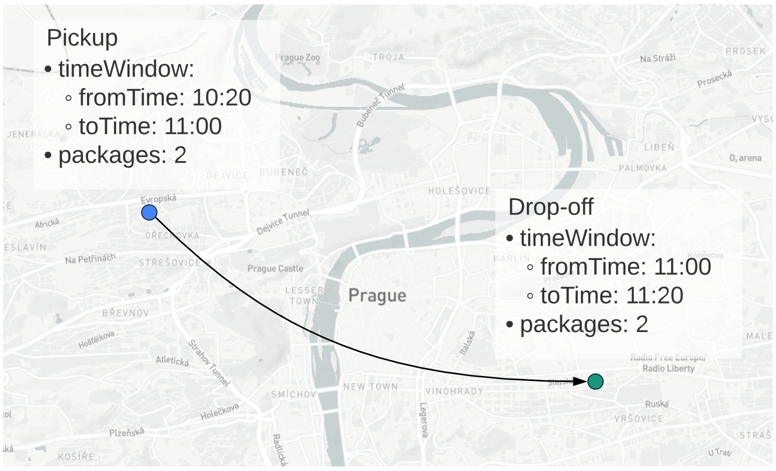

Figure 2.1: An example of a single request.

It holds that $d \geq p$ as there must be at least the same number of drop-off nodes as pickup nodes. There may be more drop-off nodes then pickup nodes as the planning phase runs in a dynamic environment, and thus some packages can be already picked up when the planning process is called. This will be further explained in section 2.1.1.

There are two types of requests $r \in R$, where each request r is a pair of nodes from V. It is either a full request $r_{i,j}^{\text{full}}$ which consists of transporting a package from the pickup node $i \in P$ to the delivery node $j \in D$, or a partial request $r_{k,i}^{\text{partial}}$ which consists of delivering an already picked up package by the driver $k \in K$ to the drop-off node $i \in D$.

A nonnegative pickup/drop-off service time t_i^{service} is associated with every node $i \in V$, which denotes the required time the driver needs to spend at node i. It is assumed that $t_i^{\text{service}} = 0 \, \forall i \in S$. A time window $w_i = [w_i^s, w_i^e]$ is associated with each node $i \in V$, where w_i^s is the start of the time window and w_i^e is the end of the time window. It is assumed that $w_i = (-\inf, \inf) \, \forall i \in S$. Finally, a travel time $t_{i,j}^{\text{travel}}$ and distance $m_{i,j}$ are associated with each pair of vertices $(i, j) \in A$. If a driver location is not known in advance, we assign $t_{k,i}^{\text{travel}} = 15$ min and $m_{k,i} = 10$ km for the driver k, assuming that it takes approximately 15 minutes and 10 kilometers to travel from a random location to another random location in the city.

Additionally, the capacity constraints are defined for the vehicles. Each driver $k \in K$ has a capacity $c_k \in \mathbb{N}$. Also, a demand $d_i \in \mathbb{Z}$ is assosiated with each node $i \in V$, and denotes how much of the capacity is utilized when completing service at node i. It holds that $d_i > 0 \, \forall i \in P$ and $d_i < 0 \, \forall i \in D$.

The output of the algorithm is a set of routes $E = \{e_0, \cdots, e_k\}$, where k is the number of drivers and the route e_l for each driver $l \in K$ is sequence of nodes $i \in V$ such that each route starts with a start node $l \in S$, where the driver is initially located. It also holds that for each full request $r_{i,j}^{\text{full}}$, pickup node i and drop-off node j are in the same route e, and node i precedes j in route e. Similarly, for each partial request $r_{l,i}^{\text{partial}}$ it holds that the drop-off node i is in route l, corresponding to driver l. For each node i in route e an estimated time of arrival t_i^{eta} and an estimated time of departure t_i^{etd} are associated, where $t_i^{\text{etd}} \geq t_i^{\text{eta}} + t_i^{\text{service}}$. Additionally, for the solution to be feasible, at any node, the driver cannot exceed its capacity, i.e., for each route e_l it must hold $\sum_{i=1}^{u} d_i \leq c_l \; \forall u \in \{1, \dots, |e_l|\}$

It is not guaranteed that a plan which satisfies all time window constrains always exists. Therefore, the time window constrains are considered soft, meaning they can be violated carrying a penalty cost.

There are two objectives to this problem. First, to minimize the deviation from the specified time windows:

$$\min \sum_{i \in V} \max(w_i^s - t_i^{\text{eta}}, t_i^{\text{eta}} - w_i^e)$$

Second, to minimize the total distance travelled by the drivers:

$$\min \sum_{e \in E} \sum_{i=0}^{|e|-1} m_{i,i+1}$$

The first objective provides for customer satisfaction, the other contributes to reducing the costs associated with the delivery.

2.1.1 Dynamicity

Online food delivery planning is a dynamic VRP as described in section 1.2.3. As new information is revealed during the planning horizon, this information needs to be incorporated into existing plans as soon as possible, so the drivers are able to react to this information. Most studies in this field consider new pieces of information as the events that trigger the replanning process [23]. Typically, the planning process should not take more than 2 minutes. Therefore, a strong emphasis is placed on the time complexity of the planning algorithm. Some of the events that trigger the replanning procedure are the following:

A new delivery order is placed When a customer makes an order, the pickup and drop-off nodes need to be incorporated into one of the existing plans.

The number of drivers changes When an existing driver goes off duty, we need to make sure that his assigned deliveries will be taken care of by someone else. Similarly, when a new driver goes on duty, he gets some deliveries that have already been assigned to his colleagues.

A driver is delayed The time estimates are never exact, and it may happen that a driver encounters a problem that causes his delay. In that case, some of the deliveries assigned to him should be handed to other drivers to compensate for the delay. The replanning procedure is triggered if the accumulated delay $d > d^{\text{lim}}$, where d^{lim} is the threshold.

Because the plans need to be recomputed in real time, it often happens that by the time the planning occurs, some packages have already been picked up by some of the couriers. In that case, we need to make sure that only the drop-off nodes are provided to the planning process and that the correct drivers will be handling them.

Another issue comes from the asynchronicity of planning. The planning procedure always takes the current state, i.e., the information it currently has, and takes some time to produce a solution. By the time the planning procedure ends, the current state may change, for example, some deliveries were picked up, another delivery was dropped off, a driver went off-duty, or a new delivery order arrived. When such a situation occurs, the planning procedure must react to these changes by either resolving those inconsistencies, or by rerunning the replanning procedure again.

2.2 Broader Context within GoDeliver

GoDeliver[8] is a software solution for governing deliveries developed by Cognitic[9] with the aim of making the city logistics simpler and more efficient. It provides clients with everything they need for managing their own fleet of drivers, including the driver application, real-time monitoring, automatic dispatch, and route planning. Figure 2.2 shows the driver application and the real-time tracking dashboard.

2.2.1 Features

The most important features that GoDeliver provides are:

Driver Management The mobile application available for *iOS* and *Android* is used to provide information to drivers about their tasks. It shows the address of the next pickup or drop-off location along with the options to open the navigation and to call the customer. The application tracks the driver's location and sends information about completed tasks to the GoDeliver backend.

Real-time Visualization The tracking dashboard for dispatchers and logistics managers allows to visualize deliveries, drivers, and routes, along with the estimated arrival times and delays.

[8] `https://godeliver.co/`

[9] `https://cognitic.ai/`

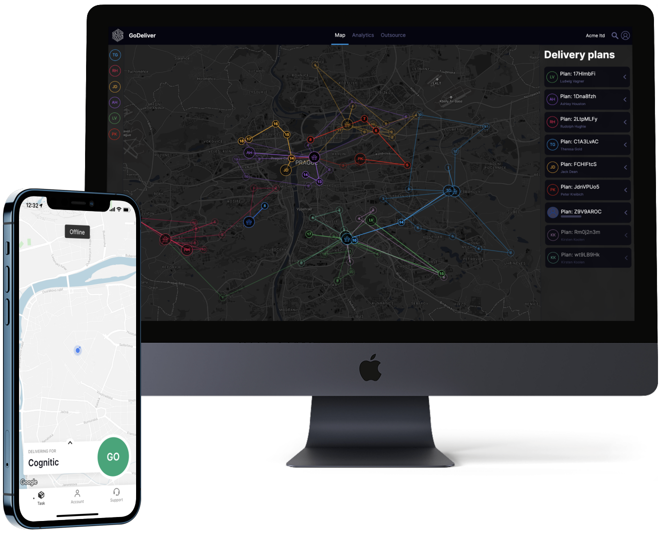

Figure 2.2: GoDeliver driver application and real-time tracking dashboard

Route Optimization Probably the most critical feature of GoDeliver is the automated route planning. This is essentially the VRP solver which supports different VRP variants, depending on the customer's needs.

Automatic Dispatch The aim of GoDeliver is to eliminate the need for human dispatchers assigning tasks to individual drivers. Therefore, automatic dispatch is a mechanism designed to tackle dynamic scenarios, where the system needs to react to new information, such as receiving or cancelling orders, changing the number of drivers, or route accidents and delays.

Customer Tracking To ensure the best possible experience for the end customers, GoDeliver offers order tracking. This web app is sent to users via SMS message or email and shows the status of their order and the current location of the driver.

Data Visualisation GoDeliver enables data-driven decisions by gathering and visualising data about the logistics fleet, such as delays, compound delivery costs, and other statistics.

External Integrations To integrate GoDeliver within existing systems, the software provides a well-documented API and plugins to popular point-of-sale systems.

2.2.2 Architecture

A simplified architecture of GoDeliver system is shown in figure 2.3. The main logic behind GoDeliver is implemented in the *Backend Service.* This service is a server application that communicates with the tracking dashboard and driver applications via a REST API. It also provides public endpoints for integrating with other systems. The data are stored in a Firestore Database[10]. On certain events that should trigger the replanning a new Celery[11] job is created. The *Replan Runner* consumes these jobs, calls the *Logistics Planner* (highlighted in yellow color), and writes the produced solution to the database.

The Logistics Planner is a stateless server that provides a single endpoint for planning. This endpoint consumes the instance of the VRP and produces a solution. In case that the solution cannot be found, an error is returned. To obtain the real distance and travel duration between each pair of nodes, the *Planning Engine* is used. This server runs our own instance of Open Source Routing Machine (OSRM)[12] which finds the shortest paths between two pairs of coordinates. This engine works over the Open Street Map[13] data that are regularly updated to include new routes and recent closures.

The Logistics Planner internally uses Google ORTools for solving the VRP instance. As mentioned in section 1.2.4, ORTools is unable to solve larger instances and is thus insufficient for real-world use cases. Therefore, this thesis aims to upgrade the planning algorithm to comply with the real-world applications.

[10] `https://firebase.google.com/docs/firestore`

[11] `https://docs.celeryproject.org/`

[12] `http://project-osrm.org/`

[13] `https://www.openstreetmap.org/`

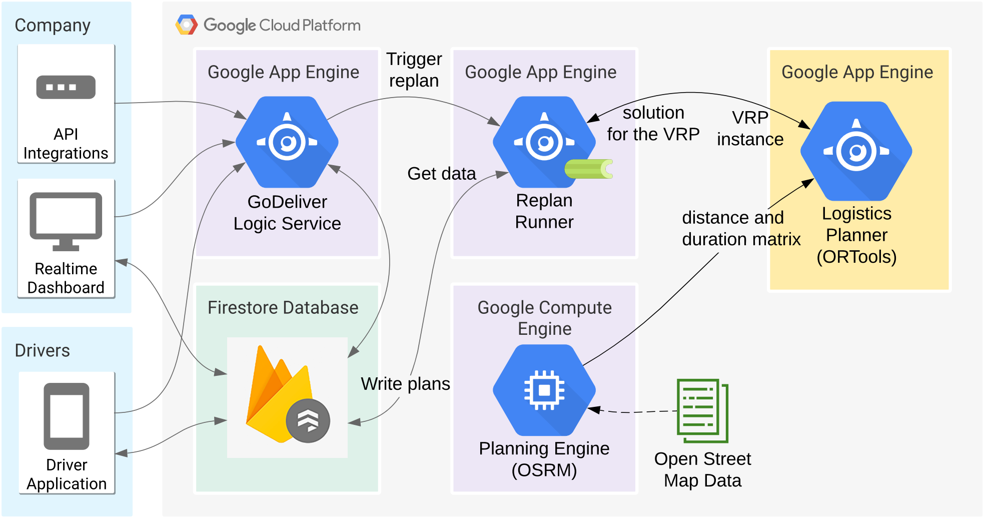

Figure 2.3: Current state of GoDeliver architecture

Chapter **3**

Methodology

Here, we propose a hybrid adaptive large neighborhood search algorithm for solving the problem described in the previous chapter. Next, we describe the methodology for evaluation of the proposed algorithm. Lastly, we illustrate the implementation of all mentioned parts.

3.1 The Algorithm for the PDPTW

The adaptive large neighborhood search (ALNS) algorithm has been successfully applied to a wide range of VRP problems, including PDPTW and DARP. In addition, the robustness of ALNS allows to efficiently solve problems with different features. However, ALNS is prone to get trapped in local optima when applied to highly constrained problems. Therefore, in this study, we combine diverse intensification strategies in the promising regions of the solution space as well as several diversification techniques to direct the search towards new and unexplored regions of the solution space. The algorithm used in this thesis is a hybrid version of ALNS, named *hybrid adaptive large neighborhood search* (HALNS) and is based on the research made on DARP by primarily Masmoudi et al. (2020) [66], and others, including Ropke and Pisinger (2006) [67], Gschwind and Drexl (2016) [63], and Masmoudi et al. (2016) [43].

The majority of the ALNS algorithms in the literature uses the following approach for restarting the search: when a new solution is not better than the current solution, and is not accepted using the acceptance criterion known from the simulated annealing algorithm, the ALNS algorithm restarts the search from a solution that is generated from the same current solution by applying the removal and insertion operators. However, our algorithm does not return to the current solution. Instead, it generates a new solution using a crossover operator known from genetic algorithms. It combines the current best solution with the new solution generated by the constructive heuristics used for obtaining the initial solution. This solution is then used as the current solution. This approach gives the algorithm more diversification power as this newly generated solution is placed in a new region of the solution space, thanks to the crossover.

```
input : Instance of PDPTW consisting of a set of requests and
        required number of drivers.
$x \leftarrow InitialSolution(instance)$ ;              /* current solution */
$x_{\text{best}} \leftarrow x$ ;                              /* best solution */
$\tau \leftarrow \tau_{\max}$ ;                                  /* temperature */
Initialize the weights of insertion, removal, and local search operators;
for $i \in 1, \ldots, i_{\max}$ do
    $x' \leftarrow$ apply removal to the current solution $x$;
    $x_{\text{new}} \leftarrow$ apply insertion to $x'$ ;            /* new solution */
    if $x_{\text{new}}$ is feasible then
        if $cost(x_{\text{new}}) < cost(x)$ or $cost(x_{\text{new}})$ satisfies acceptance
          criterion then
            $x \leftarrow x_{\text{new}}$;
        else if $cost(x_{\text{new}}) > cost(x)$ then
            $x_{\text{init}} \leftarrow InitialSolution(instance)$;
            $x \leftarrow Crossover(x_{\text{best}}, x_{\text{init}})$;
        if $cost(x_{\text{new}}) < cost(x_{\text{best}})$ then
            $x_{\text{best}} \leftarrow x_{\text{new}}$;
            $\tau_{\text{best}} \leftarrow \tau$;
        else if $cost(x_{\text{new}}) < cost(x_{\text{best}})(1 + \delta)$ then
            $x_{\text{new}} \leftarrow LocalSearch(x_{\text{new}})$;
            if $cost(x_{\text{new}}) < cost(x_{\text{best}})$ then
                $x_{\text{best}} \leftarrow x_{\text{new}}$;
                $\tau_{\text{best}} \leftarrow \tau$;
    $\tau \leftarrow \alpha\tau$;
    if $\tau < 0.01$ then
        $\tau_{\text{best}} \leftarrow 2\tau_{\text{best}}$;
        $\tau \leftarrow min(\tau_{\text{best}}, \tau_{\max})$;
    update the weights of insertion, removal and local search
      operators;
return $x_{\text{best}}$
```

Algorithm 3: Hybrid Adaptive Large Neighborhood Search for PDPTW

Additionally, in most ALNS approaches, the best solution is updated only if the newly generated solution is better than the best solution. On the contrary, our algorithm uses an acceptance function for the new solution, which works as follows: if the newly generated solution is not worse than $\delta\,\%$ from the best solution, the new solution is not discarded. Rather, this solution is intensified using the local search procedure (see section 3.1.3), and then compared with the best solution again. The better of the two solutions becomes the new best solution. This approach gives more chance for promising solutions to become new best solutions and thus results in more diversification power. On the other hand, using the local search procedure for promising solutions results in intensifying the search towards even better solutions.

These strategies form our hybrid adaptive large neighborhood search, which combines the diversification ability of the crossover procedure and the modified acceptance function with the intensification ability of the local search procedure. Although similar approaches have been used to solve the DARP by Masmoudi et al. (2020) [66], to the best of our knowledge, these novel techniques have not yet been applied to the food delivery planning problem.

The main loop of our HALNS algorithm is outlined in algorithm 3. Similarly to the ALNS known from Ropke and Pisinger (2006) [67], Gschwind and Drexl (2016) [63], and Masmoudi et al. (2016) [43], the algorithm keeps track of (1) the best solution found so far x_{best}, (2) the current solution x and (3) the newly generated solution x_{new}. The algorithm is executed for a specified number of iterations $i_{\max}$ to find the best solution x_{best} based on its cost $cost(x_{\text{best}})$ (see section 3.1.1).

In the beginning, the current solution x is initialized to the new solution obtained by the construction heuristics (see section 3.1.5). Best solution x_{best} is set to the value of current solution x. We also initialize the temperature τ to the value $\tau_{\max}$ and we initialize the weights of insertion, removal, and local search operators for the adaptive weight adjustment, further described in section 3.1.4.

In each iteration of the algorithm, the new solution x_{new} is generated from the current solution x by applying the removal and insertion operators, which are described in section 3.1.2. If the best solution x_{best} was improved in the last iteration, one removal and one insertion operator are used to generate x_{new}. Elseways, two removal operators and one insertion operator are used to further diversify the search. Our removal operators destroy the current solution by removing a number of requests, whereas the insertion operators repair the solution by embeding all unplanned requests back to the solution. The specific removal and insertion operators are selected based on their past performance. The selection mechanism is further described in section 3.1.4.

The new solution x_{new} is accepted if it is better than the current solution x or if it satisfies the acceptance criterion, i.e., we accept it with probability $e^{\left(cost(x_{\text{new}})-cost(x)\right)/\tau}$. Otherwise, a new solution is obtained by utilizing a randomly selected crossover operator (see section 3.1.6) that combines the current best solution x_{best} with a new solution obtained from the constructive heuristics x_{init} (see section 3.1.5). If the cost of the new solution $cost(x_{\text{new}})$ is lower than the cost of the current best solution $cost(x_{\text{best}})$, x_{new} becomes the new x_{best}. Else, if $cost(x_{\text{new}})$ is worse than x_{best} by a maximum of $\delta\,\%$, x_{new} is improved with the local search and becomes the new best solution x_{best} if it has lower cost compared to x_{best} after the intensification of the local search procedure.

The temperature τ, used in the acceptance criterion, is decreased in each iteration by multiplying it with the cooling rate α. If after the cooling procedure, the temperature value becomes lower than 0.01, τ_{best}, used to record the temperature when x_{best} is found, is multiplied by 2 and the value of τ is set to the value of τ_{best}. To make sure that the search does not start from scratch from a random solution, the temperature τ is limited to τ_{max}.

3.1.1 Cost Function

The cost function is an essential part of our HALNS algorithm as it determines the quality of the produced solution. As the costs of the intermediate solutions are measured several times in each iteration of the algorithm, especially during the application of insertion operators and our constructive heuristics, the cost function needs to be reasonably fast. Our objective is to minimize the divergence from the defined time windows and the total distance driven by the vehicles.

Sometimes, it may be beneficial for the driver to arrive at the customer location before the start of the time window so that he has a lower delay for the upcoming customers. However, computing the ideal times of arrival in the plan when allowing visiting the nodes before the earliest time set by the customer would increase the complexity of the whole algorithm. To overcome this issue, we do not allow to start a service on a node before its time window starts. As a result, when the driver arrives too soon at the location, he must wait until the time window starts to begin the service[14].

A solution x consists of a set of routes $E = \{e_0, \cdots, e_k\}$, where k is the number of drivers. The cost function of a single route $e \in E$ is computed as $cost(e) = \beta t_e^{\text{delay}} + \gamma t_e^{\text{distance}}$, where t_e^{delay} is the total delay of the route e in seconds, t_e^{distance} is the total travelled distance of the route e in meters, and β and γ are the global parameters of the algorithm (see section 3.1.7).

The total delay is computed as:

$$t_e^{\text{delay}} = \sum_{i=1}^{|e|-1} \max\{t_i^{\text{eta}} + t_i^{\text{service}} + t_{i,i+1}^{\text{travel}} - w_{i+1}^e, 0\}$$

where t_i^{eta} is the expected time of arrival to node i, t_i^{service} is the service time at node i, $t_{i,j}^{\text{travel}}$ is the travel time from node i to node j, and finally w_i^e is the end of the time window of node i.

[14] In practice, the driver may serve the customer sooner and use that extra time to compensate later delays, for example due to traffic.

The total distance is computed as:

$$t_e^{\text{distance}} = \sum_{i=1}^{|e|-1} m_{i,i+1}$$

where $m_{i,j}$ is the travel distance between nodes i and j.

The cost of the solution x is calculated as the sum of the costs of all routes, i.e., $cost(x) = \sum_{k \in K} cost(e_k)$, where K is the set of drivers and e_k is the route of driver k.

Figure 3.1 shows an example of traversing a single route of five nodes to compute its cost. The driver arrives at nodes 1, 3, and 5 too soon and begins the service at the start of the time window. At node 4, the driver arrives late and the difference between the end of the time window and the arrival time is the delay that is added to the total cost.

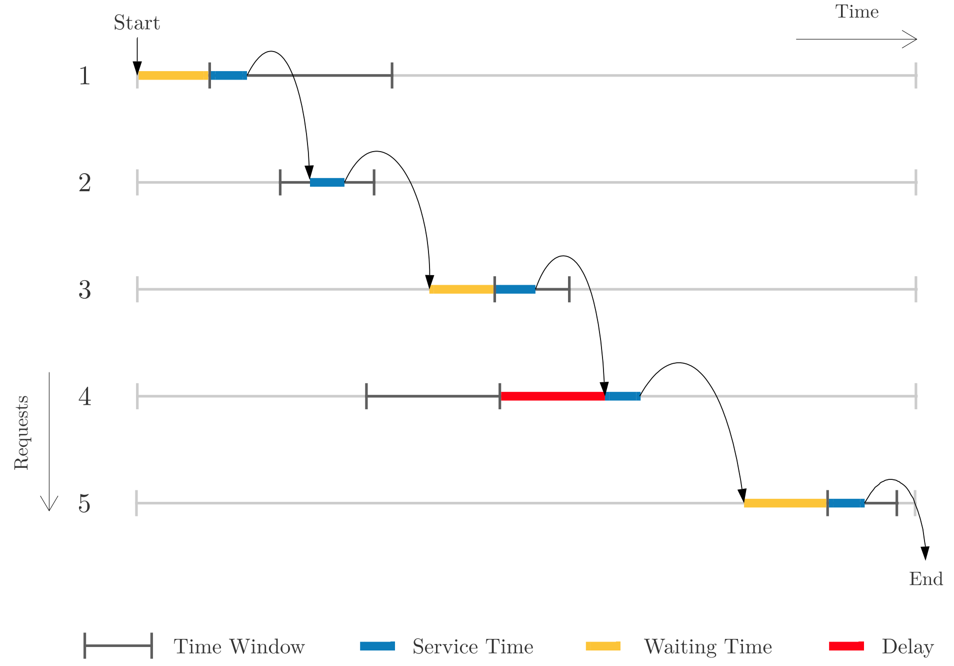

Figure 3.1: An example of a single plan traversal for computing the cost function.

3.1.2 Removal and Insertion Operators

The removal operators are used in each iteration to destroy the current solution by removing a number of requests from the solution and putting these requests into a set R. The insertion operators then repair the solution by taking the requests from the set R and inserting them back into the solution, such that the solution is feasible.

The removal and insertion operators were adopted from the existing literature, such as Ropke and Pisinger (2006) [67], Pisinger and Ropke (2007) [94], and Demir et al. (2012) [95] and adapted to our specific problem definition. The operators used in our version of HALNS are:

Random Request Removal (R1) This operator randomly selects n requests from the current solution x and removes the corresponding pickup and drop-off nodes forming a new solution x'.

Path Removal (R2) Similar to the removal operator presented by Demir et al. (2012) [95], our path removal operator randomly selects one request r from the current solution x and removes n nodes in the plan between the pickup node p_r and drop-off node d_r of request r. We also remove all corresponding nodes such that we always remove the whole request. In other words, when only a pickup node is to be removed, we also remove the drop-off node of the same request even if it is outside of the p_r and d_r path.

Related Removal (R3) This operator, based on Ropke and Pisinger (2006) [67], randomly selects one request r in the current solution and removes n most related requests. The function which defines how much two requests i and j are related is defined as follows: $R(i,j) = m_{p_i,p_j} + m_{d_i,d_j} + \rho(|t^{\text{eta}}_{p_i} - t^{\text{eta}}_{p_j}| + |t^{\text{eta}}_{d_i} - t^{\text{eta}}_{d_j}|)$, where p_i and p_j (or d_i and d_j) are the pickup nodes (or the drop-off nodes) of requests i and j, respectively; $m_{i,j}$ is the distance between two nodes; t^{eta}_i is the expected time of arrival to the node i in the current plan; and $\rho \in [0,1]$ is a control parameter. The operator then sorts the unplanned requests based on the function R and removes n requests in this order forming a new solution x'.

Time-oriented Removal (R4) This operator is a special case of the related removal operator R3, where the requests that are serviced in similar times are removed. In this case, the relatedness function is defined only by the arrival times in the current solution: $R(i,j) = |t^{\text{eta}}_{p_i} - t^{\text{eta}}_{p_j}| + |t^{\text{eta}}_{d_i} - t^{\text{eta}}_{d_j}|$.

Distance-oriented Removal (R5) This one is another special case of the related removal operator R3, where the requests in the same area as the randomly chosen request are removed. The relatedness function is defined by the distance between the pickup and drop-off nodes of the requests i and j: $R(i,j) = m_{p_i,p_j} + m_{d_i,d_j}$.

Best Position Intra-route Insertion (I1) This operator takes the unplanned requests from the set R in a random order. For each request $r \in R$, the pickup and drop-off nodes are inserted in the first route where the insertion does not violate the feasibility. The pickup and drop-off nodes are inserted in the best possible location within the route, i.e., the cost of the insertion is the lowest possible. This is determined by checking all possible combinations of insert locations while respecting the precedence constraints. This process is repeated until R is not empty.

Best Position Inter-route Insertion (I2) This operator is similar to the intra-route operator I1, but in this case, the request $r \in R$ is inserted in the best position across all routes, rather than a single route.

Sorting Time Insertion (I3) This operator is analogous to the intra-route operator I1, but the requests are first sorted by the start of their drop-off time window in an ascending order.

Greedy Insertion (I4) This operator was proposed by Ropke and Pisinger (2006) [67] and is an extension of our inter-route operator (I2). Here, the requests are selected based on the insertion cost, i.e., the request that is the cheapest to insert into the solution is inserted first, and in the best possible location. This operator adds another order of time complexity, compared to the I2 operator, because to determine the insertion cost of request r, we need to check all possible combinations of pickup and drop-off node insertions in the solution. A known problem with this heuristic is that it postpones the insertion of expensive requests to the end, where there is not much space available.

In the case of removal operators, the number of requests to remove n is selected from an uniform random distribution $\mathcal{U}(u_{\min}k, u_{\max}k)$, where k is the number of all requests in the instance.

3.1.3 Local Search Procedure

To enhance the quality of solutions, three local search operators are used. These are inspired by the existing literature and adapted to our problem:

Intra-route relocate operator (L1) This operator, adopted from Savelsbergh (1992) [96], operates on a random route in the solution. For each request r in the route it removes the pickup r_p and drop-off nodes r_d and reinserts them back into the route in the cheapest fashion, similarly to the insertion intra-route operator I1.

Inter-route relocate operator (L2) This operator is a special case of the previous operator L1, where the best insertion position is searched for in all plans in the solution, rather than the same route. Thus, this operator relocates requests between routes.

2-opt operator (L3) This operator, inspired by Lin (1965) [97], takes a random route from the solution and two positions p and q within the plan. It then reverses the order of the nodes between the points p and q. Because this operation might break the precedence constraints, a fixing procedure is then applied. This procedure iterates through the edited plan and swaps all pairs of pickup and drop-off nodes that are not in the correct order. All combinations of positions p and q are tested and the resulting solution with the lowest cost is returned.

The local search operators can only improve the current solution and cannot generate a new solution with higher cost. During the search, these operators are selected based on the first improvement strategy, which works as follows: The current operator is applied repeatedly until no further improvements are possible, then the next operator is applied. When all local search operators were used and the solution is no longer improved, the procedure ends and the current solution is returned.

In addition, the specific local search operators are selected based on their past performance using a roulette wheel mechanism, which is further described in section 3.1.4.

3.1.4 Adaptive Weight Adjustment

In section 3.1.2 we defined five removal operators and four insertion operators. Additionally, section 3.1.3 defined three local search operators. Similarly to other ALNS algorithms from the literature, we propose to use all these operators during the search. The reason behind this is that while one specific operator might be well suited to one type of instance, others might perform better on different types of instances. As a consequence, alternating between these operators results in a more robust algorithm. The adaptive weight adjustment procedure defined in this section is inspired by Ropke and Pisinger (2006) [67].

The operators are selected according to their past performance during the search. Each operator in its respective category is assigned a weight and the specific operator is selected in each iteration using a roulette wheel mechanism. The whole search is divided into *segments*, which is the number of iterations of the algorithm; here we define the segment as 100 iterations ($n_{seq} = 100$).

The probability of choosing an operator d in iteration t is given by: $P_d^t = P_d^{t-1}(1 - r_p) + r_p \pi_d / \omega_d$, where r_p is the roulette wheel parameter, π_d is the score of the operator d in the last segment, and ω_d is the counter of how many times the operator d was used in the last segment. The initial probabilities P_d^0 are constants defined in section 3.1.7.

The scores of the operators are set to 0 at the beginning of each segment. They are increased in each iteration depending on their performance: we increase the score by π_1 if the operator finds a new best solution x_{best}, by

π_2 if the operator improves the current solution x, and finally by π_3 if the operator finds a feasible solution that is worse than the current solution x but is accepted via the acceptance criterion. At the end of the segment, the weights are adjusted based on the values defined above and the counts ω are set to 0;

The reasoning behind π_1 is clear: when an operator finds a new best solution x_{best}, it has done well. Similarly, if the operator finds a solution that is accepted by the acceptance criterion, it is also successful as it advances the search. We distinguish between the situations which correspond to the parameters π_2 and π_3 as we prefer the operators that improve the solution, however, we are also interested in diversifying the search.

3.1.5 Initial Solution

The constructive heuristic that provides the initial solution at the beginning of the algorithm, and later for the crossover operation is an *insertion heuristic* adopted from Lu and Dessouky (2006) [46]. The algorithm 4 shows how the insertion heuristic works in our context. The functionality of this algorithm is analogous to the inter-route insertion operator I2, defined in section 3.1.2, but in this case, all requests are to be inserted into an initially empty solution.

```
input : The set of unplanned requests $R$ and the list of drivers $K$
shuffle the requests in $R$;
for request $r \in R$ do
    for driver $k \in K$ do
        let $e_k$ be a current route of driver $k$;
        let $l$ be the length of route $e_k$ for $i \in 1, \dots, l+1$ do
            for $j \in i+1, \dots, l+2$ do
                $e'_k \leftarrow$ insert pick-up(r) before position $i$ in plan $e_k$;
                $e_k^{i,j} \leftarrow$ insert drop-off(r) before position $j$ in plan $e'_k$;
    $k^*, i^*, j^* \leftarrow \underset{k,i,j}{arg\,min}\; cost(e_k^{i,j}) - cost(e_k)$ subj. to $e_k^{i,j}$ is feasible;
    request $r$ is inserted into the route of driver $k^*$, pickup node
      before position $i^*$ and drop-off node before position $j^*$.;
```

Algorithm 4: Insertion heuristics for creating an initial solution for our HALNS algorithm

3.1.6 Diversification Mechanism

To explore the unknown regions of the solution space and thus provide the algorithm with more diversification capability, the crossover mechanism described above is applied during the search. The operation combines the current best solution x_{best} with an initial solution x_{init} constructed by the constructive heuristic (see section 3.1.5). The intention of this operation is to find approximately the same quality solution that is placed in a different region of the solution space. Three different crossover operators that are well-known from the literature on genetic algorithms are used. The visualisation of the usage of these operators on an example route is shown in figure 3.2.

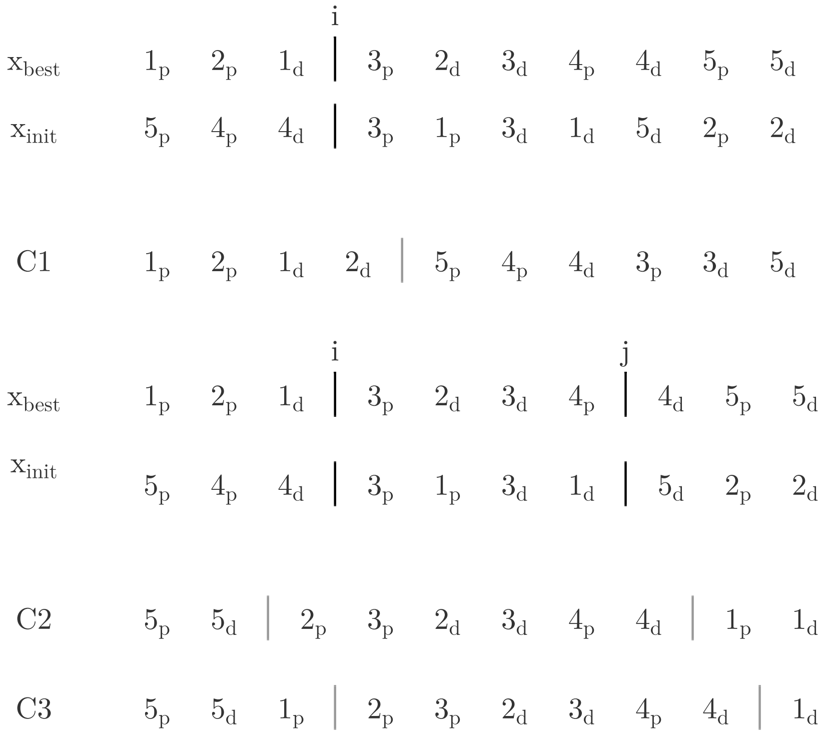
i
x_best 1p 2p 1d | 3p 2d 3d 4p 4d 5p 5d
x_init 5p 4p 4d | 3p 1p 3d 1d 5d 2p 2d
C1 1p 2p 1d 2d | 5p 4p 4d 3p 3d 5d
i j
x_best 1p 2p 1d | 3p 2d 3d 4p | 4d 5p 5d
x_init 5p 4p 4d | 3p 1p 3d 1d | 5d 2p 2d
C2 5p 5d | 2p 3p 2d 3d 4p 4d | 1p 1d
C3 5p 5d 1p | 2p 3p 2d 3d 4p 4d | 1d

Figure 3.2: A visualisation of the three crossover operators.

One-point Crossover (C1) This operator is inspired by Prins (2004) [79], who applied this operator to instances of classical VRP. In our usecase of PDP, the operator works as follows: first, a random position i is selected, such that $0 < i < \max_{k \in K} |e_k|$, where K is the set of drivers, e_k is the route of driver k, and $|e_k|$ is its length. Second, all requests whose node appears before position i in each route of x_{best} are copied to the new solution in their respective order. Finally, the rest of the requests are copied from x_{init} in the respective order, skipping the requests that are already in the new solution.

Two-point Crossover (C2) A similar operator was proposed by Goldberg and Holland (1988) [98]. In our case, two positions i and j are selected, such that $0 < i < j < \max_{k \in K} |e_k|$. The operator copies all requests, whose nodes appear between positions i and j in routes from x_{best} to the new solution while maintaining their order. Finally, the same operation is performed on nodes that appear before position i and after position j in routes from x_{init}, but in reverse order.

Linear Two-point Crossover (C3) This operator, proposed by Sevaux and Dauzère-Pérès (2003) [99] is similar to the two-point crossover operator C2. The only difference is that after copying the nodes from x_{best}, the rest of the nodes are copied from x_{init} in their respective order. After that, to ensure the precedence constraints are satisfied, a repair procedure which adds the nodes of the incomplete requests is performed.

3.1.7 Parameter Selection

The parameters mentioned throughout the algorithm description are summarized and explained in this section. Generally, the parameters are selected based on the proposals and experimental results from the literature, such as Ropke and Pisinger (2006) [67], Demir et al. (2012) [95], Leung et al. (2013) [100], and Masmoudi et al. (2016, 2020) [43, 66]. Three parameters were not adopted from the literature and are based on our experiments. These are the maximum number of iterations $i_{\text{max}} = 100\,000$, in which we observed that the solution is no longer improving on our evaluation dataset; and the coefficients of total delay $\beta = 1$ and total distance $\gamma = 0.5$ in the cost function, which seem to be a good balance between prioritizing the user satisfaction and minimizing the operational cost. Table 3.1 summarizes the parameters used in our algorithm.

3.2 Methods for Evaluation

To evaluate the performance of the HALNS algorithm described in the previous section, three important components were needed. Firstly, we created datasets with different instances of the problem, i.e., with different number of requests and drivers. Secondly, we constructed a baseline algorithm for comparison with the newly developed HALNS algorithm. Lastly, we defined the metrics to be used for evaluating both the baseline and the HALNS algorithms.

3.2.1 Evaluation Datasets

Evaluation datasets contain different sizes of the problem instances. To create these datasets, we used real data from a selected company, which delivers food from seven restaurants in Prague. The data that could match the order details to specific customers were removed to keep anonymity.

Not.	Description	Value	Source
$i_{\max}$	Maximum number of iterations	100 000	
β	Coefficient of delay in cost calculation	1	
γ	Coefficient of distance in cost calculation	0.5	
n_{seq}	Num. of iterations to update the weights	100	[43]
u_{min}	Min. % of requests removed at each iteration	0.175	[94]
$u_{\max}$	Max. % of requests removed at each iteration	0.35	[94]
r_p	Roulette wheel parameter	0.7	[43]
P_r^0	Initial probability of removal operators	0.1	[43]
P_i^0	Initial probability of insertion operators	0.125	[43]
P_{ls}^0	Initial probability of local search operators	0.125	[43]
π_1	Score of a new best solution	15	[66]
π_2	Score of a new current solution	10	[66]
π_3	Score of a feasible non-improving solution	5	[66]
$\tau_{\max}$	Initial temperature	25	[100]
α	Cooling rate	0.99975	[67, 95]

Table 3.1: The parameters used in our HALNS algorithm

For each request, the collected data relevant to our purposes are the following:

1. Pickup Location
2. Drop-off Location
3. Pickup Time Window
4. Drop-off Time Window
5. Order Creation Time
6. Number of Packages

To create the different instances, we took all requests from this company since the 1st of January 2020 until the 5th of May 2020, totalling 7 100 requests. Next, all time windows of these requests were shifted to a single day. Figure 3.3

shows a distribution of the coordinates on the map of Prague. The distribution of the starts of the time windows at drop-off locations of these requests is shown in figure 3.4. It is clear that the vast majority of all deliveries should be delivered between 11 AM and 1 PM.

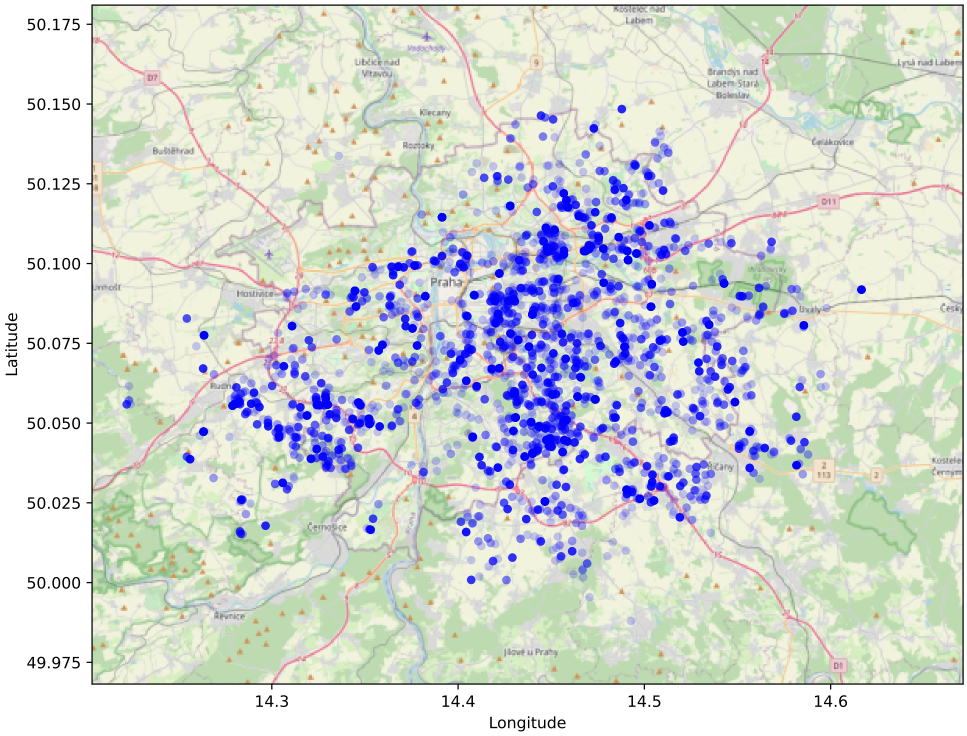

Figure 3.3: A distribution of drop-off locations in the full request dataset of 7 100 requests.

The individual instances were created by selecting random subsets of different sizes from this set. We used five different instance sizes and prepared 10 instances of each size. For each of these instance sizes, the number of drivers was determined. Based on the distribution of time windows in figure 3.4, most requests arrive in a period of 2.5 hours. Additionally, based on our experiments with our current ORTools planner, a single driver is able to finish 3.5 requests per hour. Thus, the number of drivers is computed as round$(\frac{|r|}{2.5\,3.5})$, where $|r|$ is the number of requests in the instance.

To sum up, 10 different random instances of each of the following sets were prepared. They are shown in table 3.2.

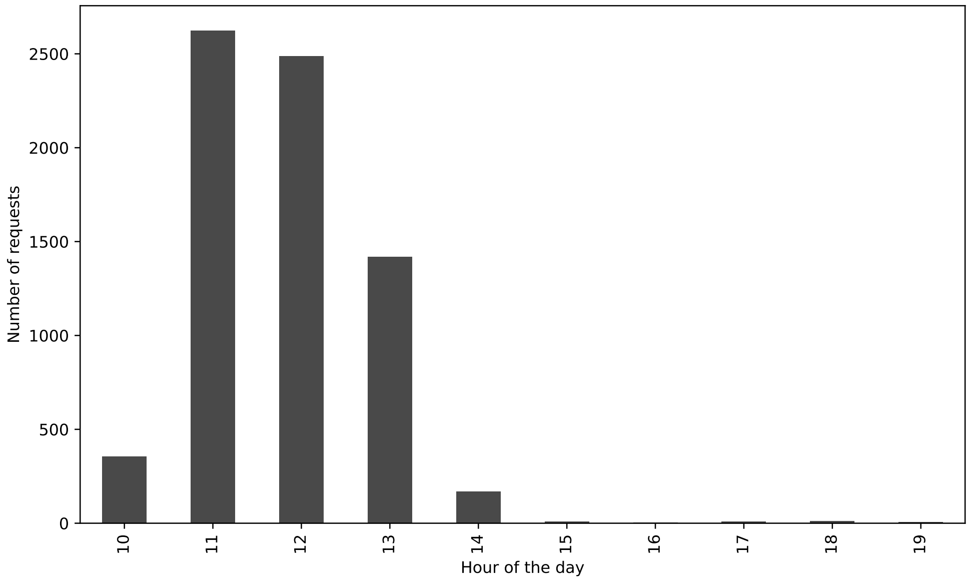

Figure 3.4: A distribution of the earliest times of drop-off set by the customers throughout the day in the full request dataset of 7 100 requests.

	Requests	**Drivers**
1	20	2
2	50	6
3	100	11
4	200	22
5	500	56

Table 3.2: 5 evaluation datasets, each containing 10 different instances

3.2.2 Baseline Algorithms

As a baseline, three methods were used. The first is a simple insertion heuristic, as was used for constructing an initial solution for the HALNS algorithm and is described in section 3.1.5. The second is the ORTools planner that is already part of the GoDeliver system and is described in section 2.2.2. The last one combines the insertion heuristic with the ORTools planner, first creating the solution using the insertion heuristic, and using that solution as an initial solution for the ORTools planner for improvement. This approach overcomes the problem that ORTools is unable to find an acceptable solution on larger instances. These three methods are visualised in figure 3.5.

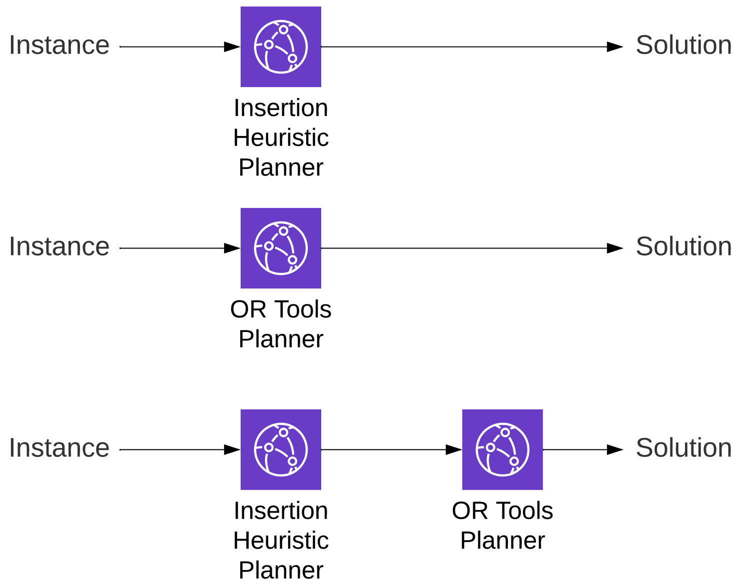

Figure 3.5: Three baseline algorithms for evaluating the performance of our HALNS algorithm.

3.2.3 Metrics

For problem instances of more than a few tens of requests, it is nearly impossible to find the optimal solution. In addition, the cost of the solution alone does not reveal the full information about the solution quality. For these reasons, we defined the metrics that help us evaluate the solutions produced by different algorithms. These metrics indicate how good the produced solutions are with respect to our objectives.

These metrics include:

Delay of Drop-off (M1) This is probably the most important metric that best describes the customer satisfaction constraint. It indicates the time difference between the drop-off time window and the estimated time of arrival of the drivers. We measure the average and the maximum delay per request.

Total Distance Travelled (M2) This is the second most important metric which shows the optimality of the routes with respect to the cost of delivery. It is defined as the sum of the total distance travelled by all drivers.

Total Time Spent (M3) This metric is an addition to the previous metric M2. It indicates the total time spent on the route across all couriers. We assume a high correlation between this metric and the M2 metric, since the distance is directly proportional to the driving time. However, this metric also includes the waiting times at the nodes and thus might produce different results.

Delivery En Route Time (M4) This metric shows the duration of a single delivery between its pickup and drop-off estimated times. In other words, it shows how long is the delivery (in our case the food) in the vehicle before it is delivered. This is especially important for perishables, such as ready-to-eat meals.

Delivery Load (M5) This metric indicates how well the algorithm is able to stack the requests within the route, i.e., it is the average number of orders picked up by the driver at one location.

3.3 Implementation

This section describes the implementation details of the planning algorithm and its integration into GoDeliver pipeline. It also briefly illustrates the developed evaluation environment that is also part of this thesis.

3.3.1 Planning Algorithm

The main HALNS algorithm described in section 3.1 as well as the insertion heuristics baseline mentioned in section 3.2.2 were implemented in *Go* language[15] version *1.15.8*. We opted for Go primarily for these reasons: it is a statically typed, compiled language and thus much more performant compared to, for example, Python; it has great support for concurrency which allows to speed up the algorithm even more; it is easy to create compiled binaries for all operating systems which makes the continuous integration pipeline easy; and finally there is a large community around Go which contributes to our confidence in its further development and support.

The most critical entity in the whole algorithm is the `solution` structure that is constantly being destroyed by removal operators, repaired by insertion operators, improved by the local search procedure, and combined with a new solution by applying the crossover mechanism. The solution instance keeps the list of n `plans`, where n is always the required number of drivers, and a set of unplanned `requests`, i.e., the requests that are not part of the solution, for example, after the removal operator is applied. Each plan contains a list of `actions`. Actions represent the graph nodes of a specific type. An action can be of three types: start, pickup, and drop-off. Each request contains

[15] `https://golang.org/`

one or two actions, depending on whether the request is already picked up before the planning procedure starts (see section 2.1.1). In the static variant, a single request contains exactly two actions: pickup and drop-off. For better understanding, figure 3.6 reveals the relationships between these entities.

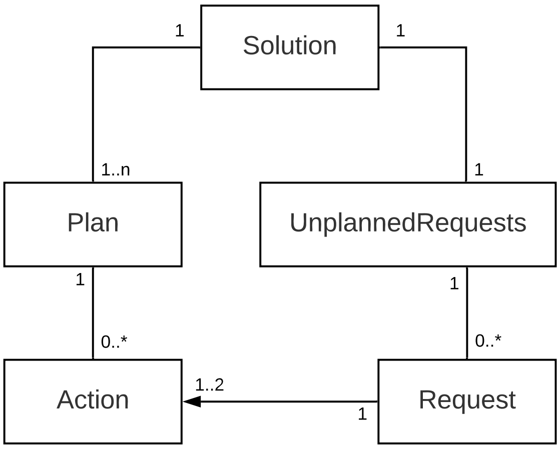

Figure 3.6: A simplified entity relationship diagram of the data model used in Go implementation.

Because Go works with the concept of interfaces rather than inheritance, so typical for other languages, the architecture of the algorithm is notably shaped by this concept. We define four interfaces for the operators used: removal, insertion, local search, and crossover. Each of these interfaces defines a different signature for the `Apply` method, that applies the operator and produces a new solution. For example, the removal operators accept two parameters: the current solution and the number of requests to remove, whereas the crossover operators need to be provided with two solutions as parameters. These interfaces are shown in figure 3.7.

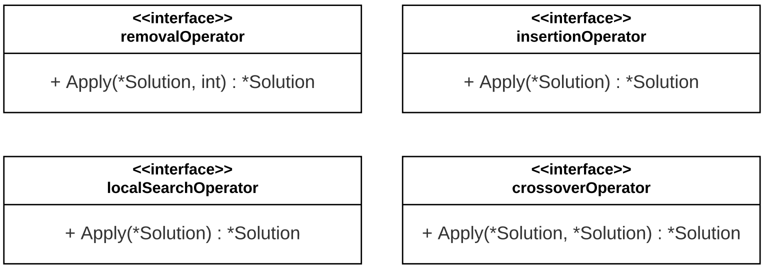
<<interface>>
removalOperator
+ Apply(*Solution, int) : *Solution
<<interface>>
insertionOperator
+ Apply(*Solution) : *Solution
<<interface>>
localSearchOperator
+ Apply(*Solution) : *Solution
<<interface>>
crossoverOperator
+ Apply(*Solution, *Solution) : *Solution

Figure 3.7: Four operator interfaces used in the Go implementation of the HALNS algorithm.

3.3.2 Integration into GoDeliver Pipeline

Extensive work was done on the integration part. The previous state of the GoDeliver Planner Service is described in section 2.2.2. The service was tightly coupled with the ORTools solver and was not extendable nor easily testable. Therefore, we abstracted the individual planning algorithms (baseline and HALNS) into their respective classes, which extend the `AbstractPlanner` class. This class is now responsible for serializing the instance, calling the respective planner, and deserializing the solution. Since the HALNS and insertion heuristics planners are implemented in Go, the GoDeliver Planner Service loads an already compiled binary as a C library, which can be called from Python. The class diagram of `AbstractPlanner` class and its subclasses is shown in figure 3.8.

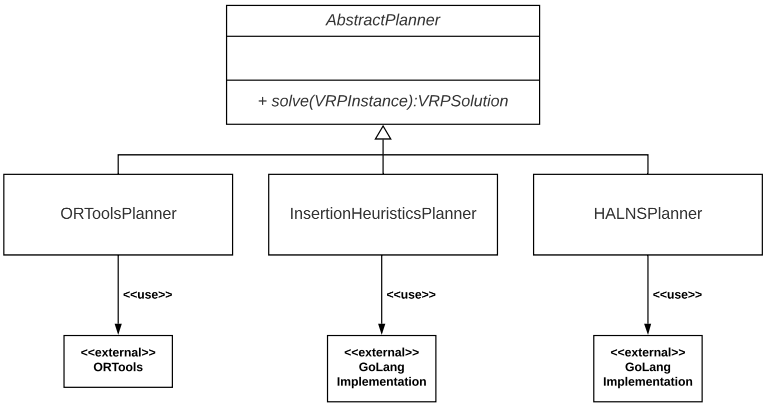
AbstractPlanner
+ solve(VRPInstance):VRPSolution
ORToolsPlanner
InsertionHeuristicsPlanner
HALNSPlanner
<<use>>
<<use>>
<<use>>
<<external>> ORTools
<<external>> GoLang Implementation
<<external>> GoLang Implementation

Figure 3.8: The Abstract Planner class, and three specific planner implementations in GoDeliver Planner.

In addition to this abstraction, we introduced a way to easily benchmark these planning algorithms and visualize the produced solutions and metrics. For that, we use *plotly*[16] framework for visualizing the plans and *streamlit*[17] framework that displays the metrics of all methods in a simple and understandable way. Figure 3.9 displays the improved and extensible architecture of GoDeliver Planner Service.

[16] `https://plotly.com/`

[17] `https://streamlit.io/`

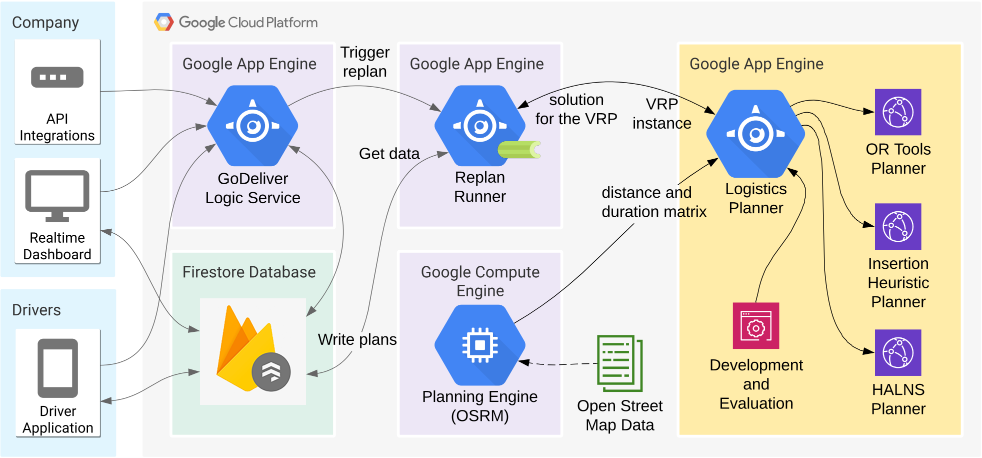

Figure 3.9: The new architecture of GoDeliver Planner (highlighted by the yellow rectange).

Chapter **4**

Results

This chapter shows the experimental results of our proposed HALNS algorithm in comparison with the defined baseline methods. It also analyzes the adaptive weight adjustment and details the parameter selection process.

4.1 Experimental results

Here, we compare our proposed HALNS algorithm described in section 3.1 with the three baseline methods detailed in section 3.2.2. The experiments were conducted on the benchmark datasets, which are illustrated in section 3.2.1. Each dataset consists of 10 instances with the same number of requests and drivers. Each experiment on each instance was executed 10 times and the results were averaged. All experiments were conducted on a machine with a 2,6 GHz 6-Core Intel Core i7 processor and 16 GB of RAM, and the maximum running time was restricted to 10 minutes. The ORTools baseline method is unable to reasonably solve instances containing more than 20 requests. Therefore, the ORTools method was evaluated only on the smallest instance containing 20 requests.

For comparison of the algorithms, we use the metrics M1 to M5 introduced in section 3.2.3 rather than the objective value. The reason is that these metrics provide much deeper insight into the quality of the produced solutions. Table 4.1 contains the experimental results in the context of the M1 metric, which indicates the average (and maximum) delay per single request at its drop-off location. Similarly, Table 4.2 highlights the M2 metric results, which indicates the average (and maximum) distance driven by each driver. In both tables, the minimum values are in bold to feature the best algorithm per each dataset. All five metrics M1 to M5 are then visualized in figure 4.2. For each metric, we show the average value per each dataset and method along with the standard deviation. In the mentioned tables and figures, *IH* stands for insertion heuristics baseline method, and *IH & ORTools* refers to the combination of insertion heuristics and ORTools, which is one of the baseline methods described in section 3.2.2.

The results in tables 4.1 and 4.2 show that the HALNS algorithm is significantly better regarding the delay and distance compared to the baseline methods on bigger instances. In contrast, on the smallest instance of 20 requests, the HALNS algorithm produces slightly worse results than insertion heuristics in combination with ORTools in terms of both metrics, however, it has slightly lower standard deviation.

From the table 4.1 we may observe much worse delays in the smallest instances, especially the maximum values. The reason behind it is the small number of drivers (two drivers used in each instance), which results in much more travelling as each driver needs to service larger areas. As a consequence, larger travel times produce larger delays.

The ORTools planner alone produces the worst solution in terms of delays on the 20 request dataset and has the largest deviation. In terms of total distance, ORTools outperforms HALNS and insertion heuristics.

The maximum values of the delay and distance shown in tables 4.1 and 4.2 suggest that HALNS is able to equally distribute the delays and distance between the requests and drivers, respectively.

There is a strong correlation between the results of metrics M2 and M3 - the distance driven and the time spent by the drivers. In terms of the M3 metric, HALNS is comparable to the IH & ORTools method. The metric M4, which denotes the average duration of travel per each request, is comparable for all methods.

The results of the metric M5, which denotes the delivery load per each pickup, show a higher ability of the HALNS algorithm to stack the orders to increase the efficiency compared to the baseline methods on all datasets.

	HALNS	**ORTools**	**IH**	**IH & ORTools**
20 r.	12.82 (**63.48**)	23.49 (98.60)	22.89 (103.31)	**9.32** (66.82)
50 r.	**0.10** (**3.49**)		1.84 (23.10)	0.67 (5.98)
100 r.	**0.04** (**1.68**)		1.18 (22.98)	0.17 (8.23)
200 r.	**0.02** (**1.62**)		0.16 (7.50)	0.09 (7.08)
500 r.	**0.07** (**6.62**)		0.14 (11.03)	0.14 (9.76)

Table 4.1: Results of metric M1 - Average (and maximum) delay in minutes at drop-off locations, evaluated on our 5 datasets, averaged from 10 runs on each instance.

We have also observed how the objective value changes with the running time. Figure 4.1 shows the objective value for each instance in the 50 request

	HALNS	**ORTools**	**IH**	**IH & ORTools**
20 r.	82.05 (88.57)	77.99 (92.96)	91.34 (97.24)	**76.88** (**81.83**)
50 r.	**46.40** (**62.30**)		56.90 (70.33)	51.14 (65.26)
100 r.	**41.30** (**58.53**)		51.90 (72.16)	44.56 (63.69)
200 r.	**35.99** (**56.87**)		43.38 (65.78)	40.80 (63.08)
500 r.	**27.98** (**53.57**)		32.81 (62.77)	32.58 (62.18)

Table 4.2: Results of metric M2 - Average (and maximum) total distance in kilometers travelled by all drivers, evaluated on our 5 datasets, averaged from 10 runs on each instance.

and 100 request datasets, which we kept running for 600 and 1 000 seconds respectively. The blue line shows the average value of the objective function of all instances. We can see that the objective value had the biggest decline within the first 10 seconds of the running time. After that, the value was changing much slower. It is important to note that the objective value at iteration 0 is the initial solution generated by the constructive heuristic.

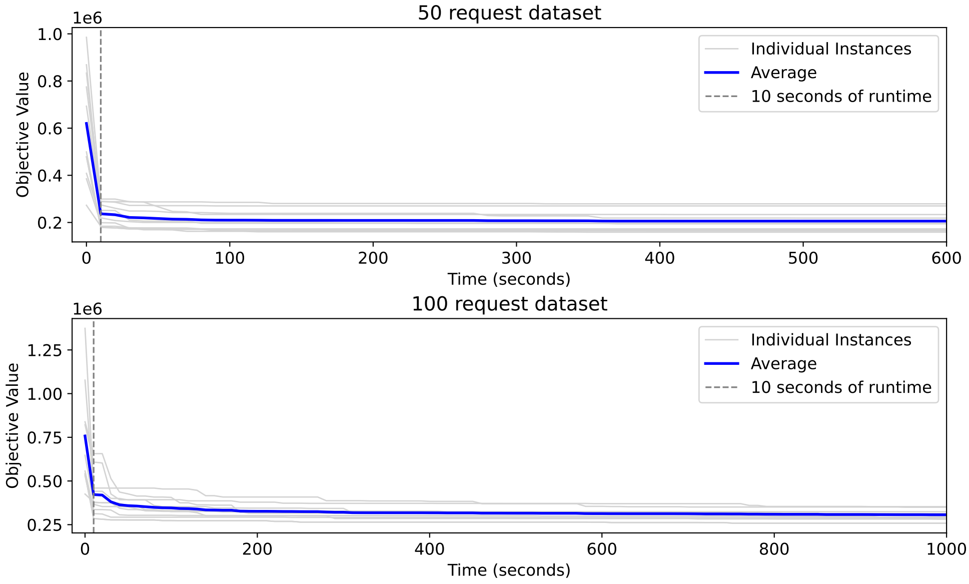

Figure 4.1: The evolution of the objective value through running time on 50 and 100 request datasets.

The objective is to minimize the total delay and the total distance by using

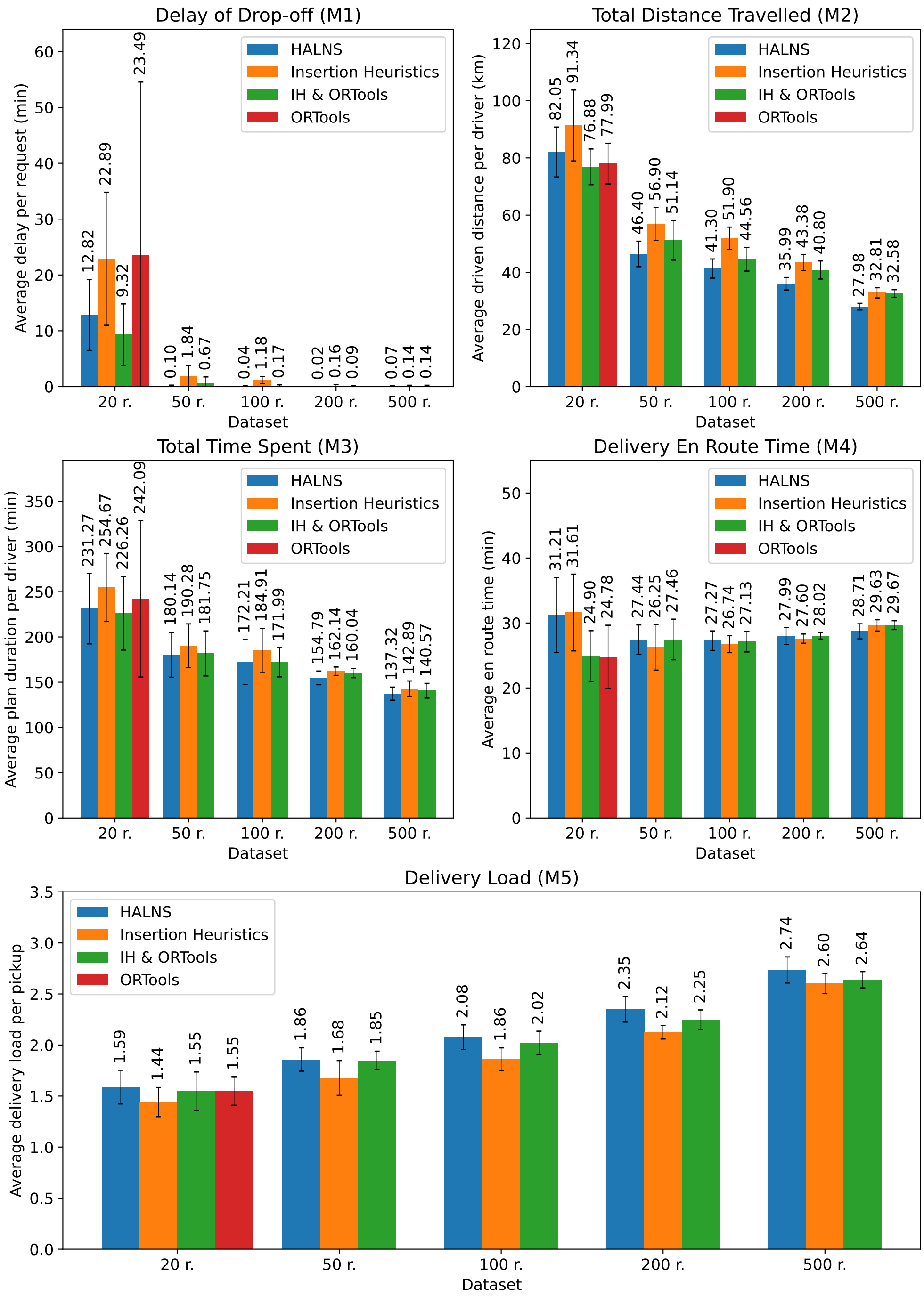

Figure 4.2: Results of metrics M1 - M5 evaluated on our 5 datasets, averaged from 10 runs of each instance in each dataset.

the formula $cost(e) = \beta t_e^{\text{delay}} + \gamma t_e^{\text{distance}}$ for each route e with the parameters $\beta = 1$ and $\gamma = 0.5$, as detailed in sections 3.1.1 and 3.1.7. Therefore, we observe the tradeoff between the total delay accumulated on all requests in the instance and the total distance travelled by all drivers. The figure 4.3 shows this tradeoff for each solution in our datasets. The figure 4.3 confirms that the HALNS algorithm produces the overall best solutions without any outliers, compared to the baseline methods.

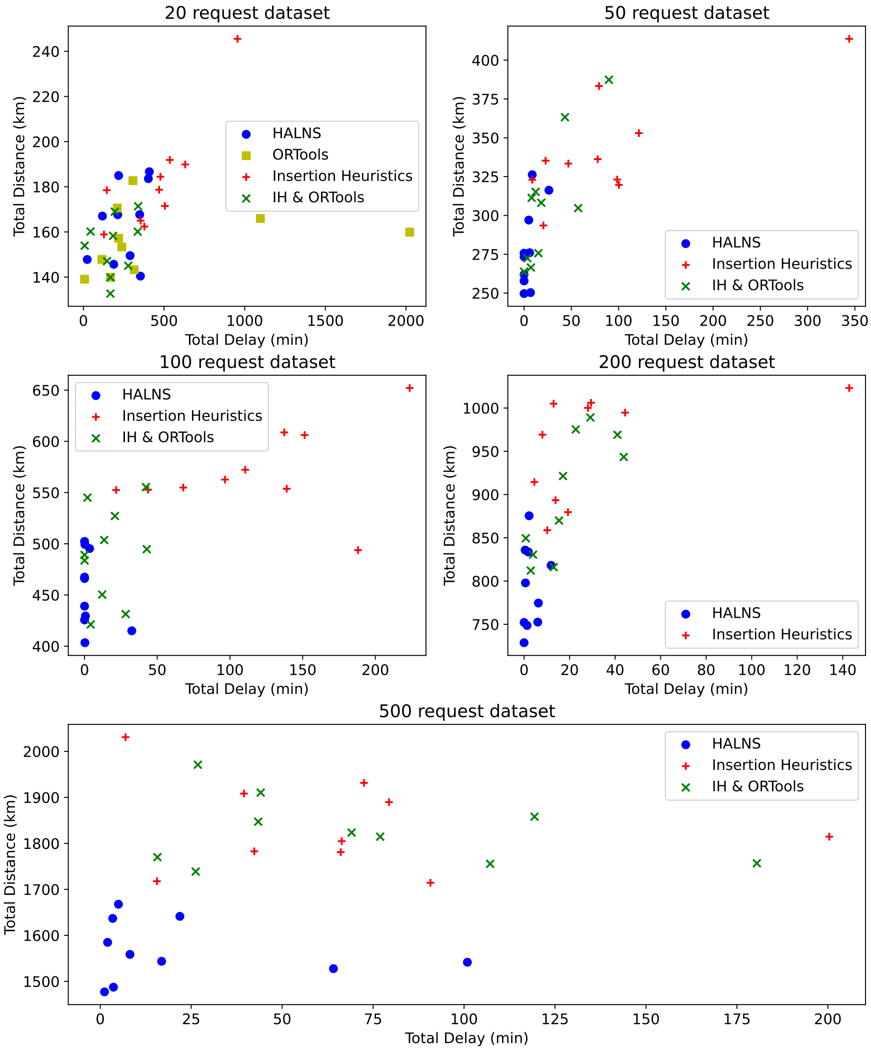

Figure 4.3: The tradeoff between the total delay and the total distance for each solution produced by the individual algorithms.

4.2 Experiments on Adaptive Weight Adjustment

This section provides some results regarding the adaptive weight adjustment procedure and details the selection process of parameters π_1, π_2 and π_3 that are used to update the weights of the operators based on their past performance, as thoroughly described in section 3.1.4.

To select the parameters π_1, π_2 and π_3 we conducted an experiment with different combinations of these parameters and compared the results. This experiment was run on a single dataset of 50 requests. It was executed 10 times and the results were averaged. The possible combinations are based on the study by Masmoudi et al. (2016) [43]. The produced solutions are in this case compared only by the objective function as only the HALNS algorithm is employed in this experiment. The results are shown in table 4.3.

Based on these results, the combination of parameters that works the best for this dataset is $(\pi_1, \pi_2, \pi_3) = (15, 5, 10)$, which was also used by Masmoudi et al. (2020) [66] and which follows the proposal of Ropke et al. (2006) [67] that $\pi_1 > \pi_3 > \pi_2$ to reinforce the diversification and thus escaping the local minima. All experiments were conducted with this set of parameters.

The performance of four insertion operators and five removal operators were studied. Figures 4.4 and 4.5 show how the weights of the operators are updated in the first 3000 iterations of the search on a single, randomly selected instance. Similarly, tables C.1 and C.2 in the appendix C show the percentage of time each insertion and removal operators were used within 5 minutes of runtime on all instances of the 100 request dataset, averaged from 10 runs of each. The results in table C.2 show that the removal operators have comparable frequencies of usage. The reason behind that may be the fact that in most cases two removal operators are applied. In contrast, the insertion operators are much more imbalanced. The operators I2 and I4 are used much more often than the other two, probably because these two operators are greedier in terms of finding the best insertion point across all routes rather than in a single route.

We also studied the complexity of the insertion and removal operators using the Go profiler. Table C.3 shows the CPU time used by each operator within 5 minutes of runtime. As expected, the greedy insertion operator I4 takes the most CPU time.

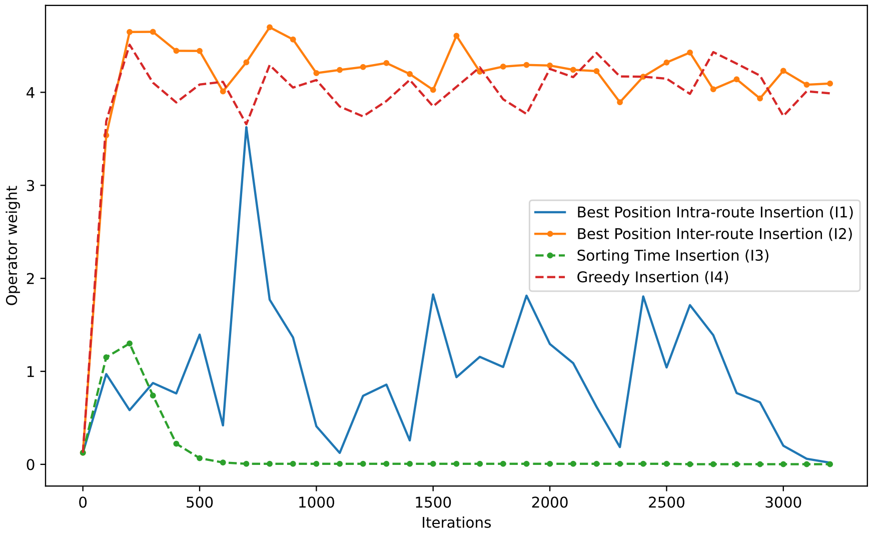

Figure 4.4: Update of the weights of our four insertion operators during the first 3 000 iterations on a single instance.

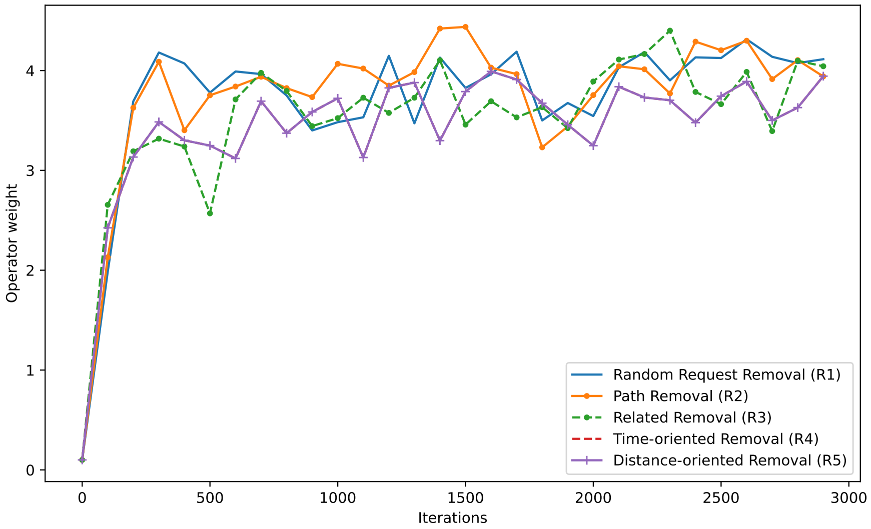

Figure 4.5: Update of the weights of our five removal operators during the first 3 000 iterations on a single instance.

Instance	**Parameters (π_1, π_2, π_3)**								
	(15, 5, 10)	(1, 10, 5)	(1, 5, 10)	(1, 5, 5)	(10, 1, 5)	(10, 5, 1)	(15, 10, 5)	(1, 1, 1)	(5, 1, 5)
1	195077	197960	195018	197029	**194462**	195633	195356	198399	195077
2	**214410**	227528	219882	219768	219882	**214410**	221973	243910	220435
3	197892	196654	196654	197996	**196016**	196654	196075	197996	197892
4	**157259**	158263	**157259**	**157259**	158263	160289	158886	**157259**	**157259**
5	270077	269402	274330	274741	274127	267039	**265941**	269740	269870
6	164381	**161878**	163186	163802	164166	**161878**	163798	163798	163927
7	278977	278977	287459	293440	280470	293440	293440	**277466**	287602
8	210639	231503	213496	**205610**	235613	206667	236569	210842	214837
9	172892	173621	**170469**	172849	172849	173098	173621	173897	173144
10	170499	173052	169367	**169296**	169870	171091	170308	170745	**169296**
Average	**203210.3**	206883.8	204712.0	205179.0	206571.8	204019.9	207596.7	206405.2	204933.9

Table 4.3: Quality of produced solutions on instances of 50 requests with different combinations of parameters π_1, π_2 and π_3. Values in bold represent the minima.

Conclusion

With the evergrowing popularity of delivery services, customers do no longer settle for a mediocre experience. This is especially true in the food delivery market, where customers demand the fastest possible delivery time for the lowest possible cost. To be able to compete in this market, companies depend upon efficient planning and order dispatching, which has to be done in real time given the dynamic environment of gastronomy.

This study contributes to the area of vehicle routing problems. We have devised a method to solve the PDPTW in the food delivery settings that uses several novel approaches adopted from the recent studies. The usefulness of our work lies in the integration of the planning algorithm to the GoDeliver system and its successful deployment to the real customers.

The proposed algorithm is adopted mainly from the research of DARP by Masmoudi et al. (2020) [66] and adapted to the PDPTW by introducing different constraints and objectives, as well as altered operators and custom parameters. The algorithm was implemented in Go language and benchmarked against insertion heuristics and Google's ORTools.

We have obtained encouraging results demonstrating that the proposed algorithm is able to solve large-scale instances emanating from real-life usecases. The results also emphasize the importance of the combination of intensification and diversification mechanisms within the search.

An important issue to address in future is regarding the performance of the algorithm. Additional performance optimization and parallelization is needed to achieve faster and more predictable execution times, which would make it easier to scale the algorithm to even larger instances. Moreover, further parameter tuning could help to find overall better solutions.

Bibliography

[1] Jan 29; 2021. US ecommerce grows 44.0% in 2020. Available from: https://www.digitalcommerce360.com/article/us-ecommerce-sales/

[2] UNCTAD. COVID-19 has changed online shopping forever, survey shows | UNCTAD. Available from: https://unctad.org/news/covid-19-has-changed-online-shopping-forever-survey-shows

[3] Statista. Revenue in the Online Food Delivery market worldwide 2024. Available from: https://www.statista.com/forecasts/891078/online-food-delivery-revenue-by-segment-worldwide

[4] Durant, D.; Dipasha, S. Online food delivery portals during COVID-19 times: an analysis of changing consumer behavior and expectations. *International Journal of Innovation Science*, 01 2021.

[5] Toth, P. *Vehicle routing: Problems, Methods, and Applications.* Philadelphia, Pennsylvania: Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104, 2015, ISBN 978-1-611973-58-7.

[6] Golden, B. *The vehicle routing problem : latest advances and new challenges.* New York London: Springer, 2008, ISBN 978-0-387-77777-1.

[7] Ropke, S. *Heuristic and exact algorithms for vehicle routing problems.* Dissertation thesis, University of Copenhagen, 01 2005.

[8] Falkner, J. K.; Schmidt-Thieme, L. Learning to Solve Vehicle Routing Problems with Time Windows through Joint Attention. 2020, 2006.09100.

[9] Dantzig, G. B.; Ramser, J. H. The Truck Dispatching Problem. *Manage. Sci.*, volume 6, no. 1, Oct. 1959: p. 80–91, ISSN 0025-1909, doi:10.1287/mnsc.6.1.80. Available from: https://doi.org/10.1287/mnsc.6.1.80

[10] Mor, A.; Speranza, M. G. Vehicle routing problems over time: a survey. *4OR*, volume 18, no. 2, Jun 2020: pp. 129–149, ISSN 1614-2411, doi:10.1007/s10288-020-00433-2. Available from: `https://doi.org/10.1007/s10288-020-00433-2`

[11] Pillac, V.; Gendreau, M.; et al. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, volume 225, no. 1, 2013: pp. 1–11, ISSN 0377-2217, doi:https://doi.org/10.1016/j.ejor.2012.08.015. Available from: `https://www.sciencedirect.com/science/article/pii/S0377221712006388`

[12] Cordeau, J.-F.; Laporte, G.; et al. Chapter 6 Vehicle Routing. In *Transportation*, *Handbooks in Operations Research and Management Science*, volume 14, edited by C. Barnhart; G. Laporte, Elsevier, 2007, pp. 367–428, doi:https://doi.org/10.1016/S0927-0507(06)14006-2. Available from: `https://www.sciencedirect.com/science/article/pii/S0927050706140062`

[13] Cordeau, J.-F.; Laporte, G.; et al. Chapter 7 Transportation on Demand. In *Transportation*, *Handbooks in Operations Research and Management Science*, volume 14, edited by C. Barnhart; G. Laporte, Elsevier, 2007, pp. 429–466, doi:https://doi.org/10.1016/S0927-0507(06)14007-4. Available from: `https://www.sciencedirect.com/science/article/pii/S0927050706140074`

[14] Bono, G. *Deep multi-agent reinforcement learning for dynamic and stochastic vehicle routing problems.* Theses, Université de Lyon, Oct. 2020. Available from: `https://tel.archives-ouvertes.fr/tel-03098433`

[15] Ibrahim, A.; Abdulaziz, R.; et al. CAPACITATED VEHICLE ROUTING PROBLEM. *International Journal of Research - GRANTHAALAYAH*, volume 7, 04 2019: pp. 310 – 327, doi:10.5281/zenodo.2636820.

[16] El-Sherbeny, N. A. Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University - Science*, volume 22, no. 3, 2010: pp. 123–131, ISSN 1018-3647, doi:https://doi.org/10.1016/j.jksus.2010.03.002. Available from: `https://www.sciencedirect.com/science/article/pii/S1018364710000297`

[17] Golden, B. *The vehicle routing problem : latest advances and new challenges.* New York London: Springer, 2008, ISBN 978-0-387-77777-1.

[18] Çağrı Koç; Bektaş, T.; et al. Thirty years of heterogeneous vehicle routing. *European Journal of Operational Research*, volume 249,

no. 1, 2016: pp. 1–21, ISSN 0377-2217, doi:https://doi.org/10.1016/j.ejor.2015.07.020. Available from: `https://www.sciencedirect.com/science/article/pii/S0377221715006530`

[19] Molina, J. C.; Salmeron, J. L.; et al. The heterogeneous vehicle routing problem with time windows and a limited number of resources. *Engineering Applications of Artificial Intelligence*, volume 94, 2020: p. 103745, ISSN 0952-1976, doi:https://doi.org/10.1016/j.engappai.2020.103745. Available from: `https://www.sciencedirect.com/science/article/pii/S095219762030155X`

[20] Asghari, M.; Mirzapour Al-e-hashem, S. M. J. Green vehicle routing problem: A state-of-the-art review. *International Journal of Production Economics*, volume 231, 2021: p. 107899, ISSN 0925-5273, doi: https://doi.org/10.1016/j.ijpe.2020.107899. Available from: `https://www.sciencedirect.com/science/article/pii/S0925527320302607`

[21] Panicker, V. V.; Mohammed, I. O. Solving a Heterogeneous Fleet Vehicle Routing Model - A practical approach. In *2018 ieee international conference on system, computation, automation and networking (icscan)*, 2018, pp. 1–5, doi:10.1109/ICSCAN.2018.8541149.

[22] Syauqi, M. H.; Zagloel, T. Y. M. Optimization of Heterogeneous Vehicle Routing Problem Using Genetic Algorithm in Courier Service. In *Proceedings of the 3rd Asia Pacific Conference on Research in Industrial and Systems Engineering 2020*, APCORISE 2020, New York, NY, USA: Association for Computing Machinery, 2020, ISBN 9781450376006, p. 48–52, doi:10.1145/3400934.3400945. Available from: `https://doi.org/10.1145/3400934.3400945`

[23] Ho, S. C.; Szeto, W.; et al. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, volume 111, 2018: pp. 395–421, ISSN 0191-2615, doi: https://doi.org/10.1016/j.trb.2018.02.001. Available from: `https://www.sciencedirect.com/science/article/pii/S0191261517304484`

[24] Belhaiza, S. A Hybrid Adaptive Large Neighborhood Heuristic for a Real-Life Dial-a-Ride Problem. *Algorithms*, volume 12, no. 2, 2019, ISSN 1999-4893, doi:10.3390/a12020039. Available from: `https://www.mdpi.com/1999-4893/12/2/39`

[25] LO, J.; MORSEMAN, S. The Perfect uberPOOL: A Case Study on Trade-Offs. *Ethnographic Praxis in Industry Conference Proceedings*, volume 2018, 10 2018: pp. 195–223, doi:10.1111/1559-8918.2018.01204.

[26] Gendreau, M.; Laporte, G.; et al. Stochastic vehicle routing. *European Journal of Operational Research*, volume 88, no. 1, 1996: pp. 3–12, ISSN

0377-2217, doi:https://doi.org/10.1016/0377-2217(95)00050-X. Available from: https://www.sciencedirect.com/science/article/pii/037722179500050X

[27] Novoa, C.; Storer, R. An approximate dynamic programming approach for the vehicle routing problem with stochastic demands. *European Journal of Operational Research*, volume 196, no. 2, July 2009: pp. 509–515. Available from: https://ideas.repec.org/a/eee/ejores/v196y2009i2p509-515.html

[28] Jaillet, P.; Wagner, M. R. Generalized Online Routing: New Competitive Ratios, Resource Augmentation, and Asymptotic Analyses. *Operations Research*, volume 56, no. 3, 2008: pp. 745–757, doi:10.1287/opre.1070.0450, https://doi.org/10.1287/opre.1070.0450. Available from: https://doi.org/10.1287/opre.1070.0450

[29] Peng, B.; Wang, J.; et al. A Deep Reinforcement Learning Algorithm Using Dynamic Attention Model for Vehicle Routing Problems. 2020, 2002.03282.

[30] Lenstra, J. K.; Kan, A. H. G. R. Complexity of vehicle routing and scheduling problems. *Networks*, volume 11, no. 2, 1981: pp. 221–227, doi:https://doi.org/10.1002/net.3230110211, https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230110211. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230110211

[31] Savelsbergh, M. W. P. Local search in routing problems with time windows. *Annals of Operations Research*, volume 4, no. 1, Dec 1985: pp. 285–305, ISSN 1572-9338, doi:10.1007/BF02022044. Available from: https://doi.org/10.1007/BF02022044

[32] Vazirani, V. *Approximation algorithms*. Berlin New York: Springer, 2001, ISBN 978-3-540-65367-7.

[33] Goyal, S. A survey on travelling salesman problem. In *Midwest Instruction and Computing Symposium*, 2010, pp. 1–9.

[34] Little, J. D. C.; Murty, K. G.; et al. An Algorithm for the Traveling Salesman Problem. *Operations Research*, volume 11, no. 6, December 1963: pp. 972–989, doi:10.1287/opre.11.6.972. Available from: https://ideas.repec.org/a/inm/oropre/v11y1963i6p972-989.html

[35] Fukasawa, R.; Longo, H.; et al. Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. *Mathematical Programming*, volume 106, no. 3, May 2006: pp. 491–511, ISSN 1436-4646, doi:10.1007/s10107-005-0644-x. Available from: https://doi.org/10.1007/s10107-005-0644-x

[36] Desrochers, M.; Desrosiers, J.; et al. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research*, volume 40, no. 2, 1992: pp. 342–354, ISSN 0030364X, 15265463. Available from: http://www.jstor.org/stable/171457

[37] KOHL, N.; DESROSIERS, J.; et al. 2-Path Cuts for the Vehicle Routing Problem with Time Windows. *Transportation Science*, volume 33, no. 1, 1999: pp. 101–116, ISSN 00411655, 15265447. Available from: http://www.jstor.org/stable/25768848

[38] Lu, Q.; Dessouky, M. An Exact Algorithm for the Multiple Vehicle Pickup and Delivery Problem. *Transportation Science*, volume 38, no. 4, 2004: pp. 503–514, ISSN 00411655, 15265447. Available from: http://www.jstor.org/stable/25769222

[39] Røpke, S.; Cordeau, J.-F.; et al. Models and a Branch-and-Cut Algorithm for Pickup and Delivery Problems with Time Windows. *Networks*, volume 49, no. 4, 2007: pp. 258 – 272, ISSN 0028-3045, doi: 10.1002/net.20177.

[40] Ropke, S.; Cordeau, J.-F. Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, volume 43, no. 3, 2009: pp. 267–286. Available from: https://EconPapers.repec.org/RePEc:inm:ortrsc:v:43:y:2009:i:3:p:267-286

[41] Marković, N.; Nair, R.; et al. Optimizing dial-a-ride services in Maryland: Benefits of computerized routing and scheduling. *Transportation Research Part C: Emerging Technologies*, volume 55, 2015: pp. 156–165, ISSN 0968-090X, doi:https://doi.org/10.1016/j.trc.2015.01.011, engineering and Applied Sciences Optimization (OPT-i) - Professor Matthew G. Karlaftis Memorial Issue. Available from: https://www.sciencedirect.com/science/article/pii/S0968090X15000133

[42] Wong, K.; Han, A.; et al. On dynamic demand responsive transport services with degree of dynamism. *Transportmetrica A: Transport Science*, volume 10, no. 1, 2014: pp. 55–73, doi:10.1080/18128602.2012.694491, https://doi.org/10.1080/18128602.2012.694491. Available from: https://doi.org/10.1080/18128602.2012.694491

[43] Masmoudi, M. A.; Hosny, M.; et al. Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem. *Transportation Research Part E: Logistics and Transportation Review*, volume 96, 2016: pp. 60–80, ISSN 1366-5545, doi:https://doi.org/10.1016/j.tre.2016.10.002.

Available from: https://www.sciencedirect.com/science/article/pii/S1366554516304070

[44] Braekers, K.; Caris, A.; et al. Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transportation Research Part B: Methodological*, volume 67, 2014: pp. 166–186, ISSN 0191-2615, doi:https://doi.org/10.1016/j.trb.2014.05.007. Available from: https://www.sciencedirect.com/science/article/pii/S0191261514000800

[45] Jaw, J.-J.; Odoni, A. R.; et al. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, volume 20, no. 3, 1986: pp. 243–257, ISSN 0191-2615, doi:https://doi.org/10.1016/0191-2615(86)90020-2. Available from: https://www.sciencedirect.com/science/article/pii/0191261586900202

[46] Lu, Q.; Dessouky, M. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, volume 175, 12 2006: pp. 672–687, doi:10.1016/j.ejor.2005.05.012.

[47] Fiedler, D.; Čertický, M.; et al. The Impact of Ridesharing in Mobility-on-Demand Systems: Simulation Case Study in Prague. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 1173–1178, doi:10.1109/ITSC.2018.8569451.

[48] Nanry, W. P.; Wesley Barnes, J. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, volume 34, no. 2, February 2000: pp. 107–121. Available from: https://ideas.repec.org/a/eee/transb/v34y2000i2p107-121.html

[49] Cordeau, J.-F.; Laporte, G. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, volume 37, no. 6, 2003: pp. 579–594, ISSN 0191-2615, doi:https://doi.org/10.1016/S0191-2615(02)00045-0. Available from: https://www.sciencedirect.com/science/article/pii/S0191261502000450

[50] Zachariadis, E. E.; Kiranoudis, C. T. A Strategy for Reducing the Computational Complexity of Local Search-Based Methods for the Vehicle Routing Problem. *Comput. Oper. Res.*, volume 37, no. 12, Dec. 2010: p. 2089–2105, ISSN 0305-0548, doi:10.1016/j.cor.2010.02.009. Available from: https://doi.org/10.1016/j.cor.2010.02.009

[51] Osman, I. H. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, volume 41, no. 4, Dec 1993: pp. 421–451, ISSN 1572-9338, doi:10.1007/BF02023004. Available from: `https://doi.org/10.1007/BF02023004`

[52] Gendreau, M.; Laporte, G.; et al. Metaheuristics for the Capacitated VRP. *Discrete Mathematics and Applications - The Vehicle Routing Problem*, 01 2001, doi:10.1137/1.9780898718515.ch6.

[53] Schneider, M.; Stenger, A.; et al. The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations. *Transportation Science*, volume 48, 03 2014: pp. 500–520, doi:10.1287/trsc.2013.0490.

[54] Mladenović, N.; Hansen, P. Variable neighborhood search. *Computers & Operations Research*, volume 24, no. 11, 1997: pp. 1097–1100, ISSN 0305-0548, doi:https://doi.org/10.1016/S0305-0548(97)00031-2. Available from: `https://www.sciencedirect.com/science/article/pii/S0305054897000312`

[55] Li, H.; Lim, A. A Metaheuristic for the Pickup and Delivery Problem with Time Windows. *International Journal on Artificial Intelligence Tools*, volume 12, no. 02, 2003: pp. 173–186, doi:10.1142/S0218213003001186, `https://doi.org/10.1142/S0218213003001186`. Available from: `https://doi.org/10.1142/S0218213003001186`

[56] Kytöjoki, J.; Nuortio, T.; et al. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, volume 34, no. 9, 2007: pp. 2743–2757, ISSN 0305-0548, doi:https://doi.org/10.1016/j.cor.2005.10.010. Available from: `https://www.sciencedirect.com/science/article/pii/S0305054805003394`

[57] Parragh, S.; Doerner, K.; et al. A heuristic two-phase solution method for the multi-objective dial-a-ride problem. *Networks*, volume 54, 12 2009: pp. 227–242, doi:10.1002/net.20335.

[58] Schilde, M.; Doerner, K.; et al. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers & Operations Research*, volume 38, no. 12, 2011: pp. 1719–1730, ISSN 0305-0548, doi:https://doi.org/10.1016/j.cor.2011.02.006. Available from: `https://www.sciencedirect.com/science/article/pii/S0305054811000475`

[59] Schilde, M.; Doerner, K.; et al. Integrating stochastic time-dependent travel speed in solution methods for the dynamic dial-a-ride problem. *European Journal of Operational Research*, volume 238, no. 1, 2014: pp. 18–30, ISSN 0377-2217, doi:https://doi.org/10.1016/j.ejor.2014.03.005.

Available from: https://www.sciencedirect.com/science/article/pii/S0377221714002197

[60] Muelas, S.; LaTorre, A.; et al. A variable neighborhood search algorithm for the optimization of a dial-a-ride problem in a large city. *Expert Systems with Applications*, volume 40, no. 14, 2013: pp. 5516–5531, ISSN 0957-4174, doi:https://doi.org/10.1016/j.eswa.2013.04.015. Available from: https://www.sciencedirect.com/science/article/pii/S0957417413002522

[61] Muelas, S.; LaTorre, A.; et al. A distributed VNS algorithm for optimizing dial-a-ride problems in large-scale scenarios. *Transportation Research Part C: Emerging Technologies*, volume 54, 2015: pp. 110–130, ISSN 0968-090X, doi:https://doi.org/10.1016/j.trc.2015.02.024. Available from: https://www.sciencedirect.com/science/article/pii/S0968090X15000790

[62] Detti, P.; Papalini, F.; et al. A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare. *Omega*, volume 70, 2017: pp. 1–14, ISSN 0305-0483, doi:https://doi.org/10.1016/j.omega.2016.08.008. Available from: https://www.sciencedirect.com/science/article/pii/S0305048316305266

[63] Gschwind, T.; Drexl, M. Adaptive Large Neighborhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem. Working Papers 1624, Gutenberg School of Management and Economics, Johannes Gutenberg-Universität Mainz, Dec. 2016. Available from: https://ideas.repec.org/p/jgu/wpaper/1624.html

[64] Shaw, P. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 1997.

[65] Azi, N.; Gendreau, M.; et al. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, volume 41, 2014: pp. 167–173, ISSN 0305-0548, doi:https://doi.org/10.1016/j.cor.2013.08.016. Available from: https://www.sciencedirect.com/science/article/pii/S0305054813002220

[66] Masmoudi, M. A.; Hosny, M.; et al. Hybrid adaptive large neighborhood search algorithm for the mixed fleet heterogeneous dial-a-ride problem. *Journal of Heuristics*, volume 26, no. 1, February 2020: pp. 83–118, doi:10.1007/s10732-019-09424-. Available from: https://ideas.repec.org/a/spr/joheur/v26y2020i1d10.1007_s10732-019-09424-x.html

[67] Ropke, S.; Pisinger, D. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, volume 40, 11 2006: pp. 455–472, doi:10.1287/trsc.1050.0135.

[68] Braekers, K.; Kovacs, A. A. A multi-period dial-a-ride problem with driver consistency. *Transportation Research Part B: Methodological*, volume 94, 2016: pp. 355–377, ISSN 0191-2615, doi:https://doi.org/10.1016/j.trb.2016.09.010. Available from: `[https://www.sciencedirect.com/science/article/pii/S0191261515301570](https://www.sciencedirect.com/science/article/pii/S0191261515301570)`

[69] Belhaiza, S. A data driven hybrid heuristic for the dial-a-ride problem with time windows. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017, pp. 1–8, doi:10.1109/SSCI.2017.8285366.

[70] Drexl, M. On the One-to-One Pickup-and-Delivery Problem with Time Windows and Trailers. Working Papers 1816, Gutenberg School of Management and Economics, Johannes Gutenberg-Universität Mainz, Oct. 2018. Available from: `https://ideas.repec.org/p/jgu/wpaper/1816.html`

[71] Belhaiza, S. A Hybrid Adaptive Large Neighborhood Heuristic for a Real-Life Dial-a-Ride Problem. *Algorithms*, volume 12, no. 2, 2019, ISSN 1999-4893, doi:10.3390/a12020039. Available from: `[https://www.mdpi.com/1999-4893/12/2/39](https://www.mdpi.com/1999-4893/12/2/39)`

[72] Wang, Y.; Lei, L.; et al. Towards delivery-as-a-service: Effective neighborhood search strategies for integrated delivery optimization of E-commerce and static O2O parcels. *Transportation Research Part B: Methodological*, volume 139, 2020: pp. 38–63, ISSN 0191-2615, doi:https://doi.org/10.1016/j.trb.2020.06.003. Available from: `[https://www.sciencedirect.com/science/article/pii/S0191261520303428](https://www.sciencedirect.com/science/article/pii/S0191261520303428)`

[73] Cauchi, M.; Scerri, K. An Improved Variable Neighbourhood Search Algorithm for Selective Dial-a-Ride Problems. In *2020 IEEE 20th Mediterranean Electrotechnical Conference (MELECON)*, 2020, pp. 652–657, doi:10.1109/MELECON48756.2020.9140695.

[74] Malheiros, I.; Ramalho, R.; et al. A hybrid algorithm for the multi-depot heterogeneous dial-a-ride problem. *Computers & Operations Research*,

volume 129, 2021: p. 105196, ISSN 0305-0548, doi:https://doi.org/10.1016/j.cor.2020.105196. Available from: https://www.sciencedirect.com/science/article/pii/S0305054820303130

[75] Masmoudi, M. A.; Braekers, K.; et al. A hybrid Genetic Algorithm for the Heterogeneous Dial-A-Ride Problem. *Computers & Operations Research*, volume 81, 2017: pp. 1–13, ISSN 0305-0548, doi:https://doi.org/10.1016/j.cor.2016.12.008. Available from: https://www.sciencedirect.com/science/article/pii/S0305054816303070

[76] Solnon, C. *Ant colony optimization and constraint programming*. London Hoboken, NJ: ISTE John Wiley, 2010, ISBN 978-1-848-21130-8.

[77] Reimann, M.; Doerner, K.; et al. D-ants: Savings Based Ants Divide and Conquer the Vehicle Routing Problem. *Computers & Operations Research*, volume 31, 04 2004: pp. 563–591, doi:10.1016/S0305-0548(03)00014-5.

[78] Schyns, M. An ant colony system for responsive dynamic vehicle routing. *European Journal of Operational Research*, volume 245, no. 3, 2015: pp. 704–718, ISSN 0377-2217, doi:https://doi.org/10.1016/j.ejor.2015.04.009. Available from: https://www.sciencedirect.com/science/article/pii/S0377221715002817

[79] Prins, C. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Computer & Operations Research 31(12), 1985-2002. *Computers & Operations Research*, volume 31, 10 2004: pp. 1985–2002, doi:10.1016/S0305-0548(03)00158-8.

[80] Parragh, S. N.; Schmid, V. Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, volume 40, no. 1, 2013: pp. 490–497, ISSN 0305-0548, doi:https://doi.org/10.1016/j.cor.2012.08.004. Available from: https://www.sciencedirect.com/science/article/pii/S0305054812001694

[81] Hottung, A.; Tierney, K. Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem. *CoRR*, volume abs/1911.09539, 2019, 1911.09539. Available from: http://arxiv.org/abs/1911.09539

[82] Potvin, J.-Y.; Dubé, D.; et al. A hybrid approach to vehicle routing using neural networks and genetic algorithms. *Applied Intelligence*, volume 6, no. 3, Jul 1996: pp. 241–252, ISSN 1573-7497, doi:10.1007/BF00126629. Available from: https://doi.org/10.1007/BF00126629

[83] Potvin, J.-Y.; Rousseau, J.-M. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, volume 66, no. 3, 1993: pp. 331 – 340, ISSN 0377-2217, doi:https://doi.org/10.1016/0377-2217(93)90221-8. Available from: `http://www.sciencedirect.com/science/article/pii/0377221793902218`

[84] Kool, W.; van Hoof, H.; et al. Attention, Learn to Solve Routing Problems! 2019, `1803.08475`.

[85] Nazari, M.; Oroojlooy, A.; et al. Deep Reinforcement Learning for Solving the Vehicle Routing Problem. *CoRR*, volume abs/1802.04240, 2018, `1802.04240`. Available from: `http://arxiv.org/abs/1802.04240`

[86] Li, J.; Xin, L.; et al. Heterogeneous Attentions for Solving Pickup and Delivery Problem via Deep Reinforcement Learning. *IEEE Transactions on Intelligent Transportation Systems*, 2021: pp. 1–10, doi:10.1109/TITS.2021.3056120.

[87] Google. Vehicle Routing Problem — OR-Tools — Google Developers. Available from: `https://developers.google.com/optimization/routing/vrp`

[88] Karkula, M.; Duda, J.; et al. Comparison of capabilities of recent open-source tools for solving capacitated vehicle routing problems with time windows. In *Proceedings CLC 2019*, 2020, ISBN 978-80-87294-96-3, pp. 72–77.

[89] VROOM-Project. VROOM-Project/vroom. Available from: `https://github.com/VROOM-Project/vroom`

[90] Bräysy, O.; Gendreau, M. Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. *Transportation Science*, volume 39, 02 2005: pp. 104–118, doi:10.1287/trsc.1030.0056.

[91] Schröder, S. java toolkit for rich VRPs and TSPs. Available from: `https://jsprit.github.io/`

[92] Li, C.; Mirosa, M.; et al. Review of Online Food Delivery Platforms and their Impacts on Sustainability. *Sustainability*, volume 12, no. 14, 2020, ISSN 2071-1050, doi:10.3390/su12145528. Available from: `https://www.mdpi.com/2071-1050/12/14/5528`

[93] Cordeau, J.-F. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Operations Research*, volume 54, no. 3, 2006: pp. 573–586, ISSN 0030364X, 15265463.

[94] Pisinger, D.; Ropke, S. A general heuristic for vehicle routing problems. *Computers & Operations Research*, volume 34, no. 8, 2007: pp. 2403–2435, ISSN 0305-0548, doi:https://doi.org/10.1016/j.cor.2005.09.012. Available from: `https://www.sciencedirect.com/science/article/pii/S0305054805003023`

[95] Demir, E.; Bektaş, T.; et al. An adaptive large neighborhood search heuristic for the Pollution-Routing Problem. *European Journal of Operational Research*, volume 223, no. 2, 2012: pp. 346–359, ISSN 0377-2217, doi:https://doi.org/10.1016/j.ejor.2012.06.044. Available from: `https://www.sciencedirect.com/science/article/pii/S0377221712004997`

[96] Savelsbergh, M. W. P. The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. *ORSA Journal on Computing*, volume 4, no. 2, 1992: pp. 146–154, doi:10.1287/ijoc.4.2.146, `https://doi.org/10.1287/ijoc.4.2.146`. Available from: `https://doi.org/10.1287/ijoc.4.2.146`

[97] Lin, S. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, volume 44, no. 10, 1965: pp. 2245–2269, doi:10.1002/j.1538-7305.1965.tb04146.x.

[98] Goldberg, D. E.; Holland, J. H. Genetic Algorithms and Machine Learning. *Machine Learning*, volume 3, no. 2, Oct 1988: pp. 95–99, ISSN 1573-0565, doi:10.1023/A:1022602019183. Available from: `https://doi.org/10.1023/A:1022602019183`

[99] Sevaux, M.; Dauzère-Pérès, S. Genetic algorithms to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research*, volume 151, 12 2003: pp. 296–306, doi:10.1016/S0377-2217(02)00827-5.

[100] Leung, S. C.; Zhang, Z.; et al. A meta-heuristic algorithm for heterogeneous fleet vehicle routing problems with two-dimensional loading constraints. *European Journal of Operational Research*, volume 225, no. 2, 2013: pp. 199–210, doi:10.1016/j.ejor.2012.09.02. Available from: `https://ideas.repec.org/a/eee/ejores/v225y2013i2p199-210.html`

Appendix A

Acronyms

ALNS Adaptive Large Neighborhood Search

API Application Programming Interface

ARP Arc Routing Problem

B&B Branch-and-Bound

CVRP Capacitated Vehicle Routing Problem

DAFP Dial-a-Flight Problem

DARP Dial-a-Ride Problem

FSMVRP Fleet Size and Mix Vehicle Routing Problem

HALNS Hybrid Adaptive Large Neighborhood Search

HFVRP Heterogenous Fleet Vehicle Routing Problem

HFVRPTW Heterogenous Fleet Vehicle Routing Problem with Time Windows

HVRP Heterogenous Vehicle Routing Problem

PDP Pickup and Delivery Problem

PDPTW Vehicle Routing Problem with Time Windows

TSP Travelling Salesman Problem

VROOM Vehicle Routing Open-source Optimization Machine

VRP Vehicle Routing Problem

VRPPD Vehicle Routing Problem with Pickup and Delivery

VRPTW Vehicle Routing Problem with Time Windows

Appendix B

Contents of an enclosed CD

Also available via https://github.com/davidmokos/halns.

```
 README.md.................................the file with the instructions
 godeliver-planner.......the folder with the GoDeliver Planner service
 halns......................the HALNS implementation in Go language
 thesis...................................the directory with the thesis
    MT_Mokos_David_2021.pdf.........the thesis text in the PDF format
    sources...........................LaTeX source codes of the thesis
```

Appendix C

Tables

Instance	Insertion Operators			
	I1	**I2**	**I3**	**I4**
1	2.23 %	46.88 %	4.32 %	46.57 %
2	12.56 %	41.48 %	6.85 %	39.11 %
3	11.68 %	43.46 %	2.24 %	42.61 %
4	1.84 %	48.54 %	2.17 %	47.45 %
5	7.24 %	46.08 %	2.29 %	44.40 %
6	4.58 %	41.37 %	11.84 %	42.21 %
7	3.98 %	46.53 %	1.45 %	48.04 %
8	8.90 %	44.18 %	5.04 %	41.88 %
9	8.13 %	42.97 %	4.54 %	44.36 %
10	4.31 %	45.59 %	6.85 %	43.24 %
Average	**6.55 %**	**44.71 %**	**4.76 %**	**43.99 %**

Table C.1: Percentage of use within 5 minutes of runtime of all insertion operators of HALNS algorithm.

Instance	Removal Operators				
	R1	**R2**	**R3**	**R4**	**R5**
1	22.05 %	19.48 %	19.17 %	19.73 %	19.57 %
2	21.37 %	19.58 %	19.46 %	20.22 %	19.37 %
3	20.36 %	19.67 %	20.47 %	20.22 %	19.28 %
4	21.05 %	21.43 %	18.90 %	21.11 %	17.51 %
5	20.43 %	20.80 %	19.45 %	20.86 %	18.47 %
6	20.16 %	20.04 %	20.07 %	21.12 %	18.60 %
7	20.43 %	21.15 %	18.56 %	20.83 %	19.04 %
8	20.17 %	20.35 %	18.88 %	21.20 %	19.40 %
9	22.10 %	20.00 %	19.00 %	19.97 %	18.93 %
10	20.82 %	21.44 %	19.39 %	19.48 %	18.88 %
Average	**20.89 %**	**20.39 %**	**19.33 %**	**20.48 %**	**18.91 %**

Table C.2: Percentage of use within 5 minutes of runtime of all removal operators of HALNS algorithm.

	Insertion Operators			
Operator	**I1**	**I2**	**I3**	**I4**
Percentage	2.82 %	25.33 %	2.21 %	**69.64** %

	Removal Operators				
Operator	**R1**	**R2**	**R3**	**R4**	**R5**
Percentage	10.02 %	12.2 %	**31.38** %	29.98 %	16.42 %

Table C.3: Percentage of CPU used by each operator of HALNS algorithm within 5 minutes of runtime.