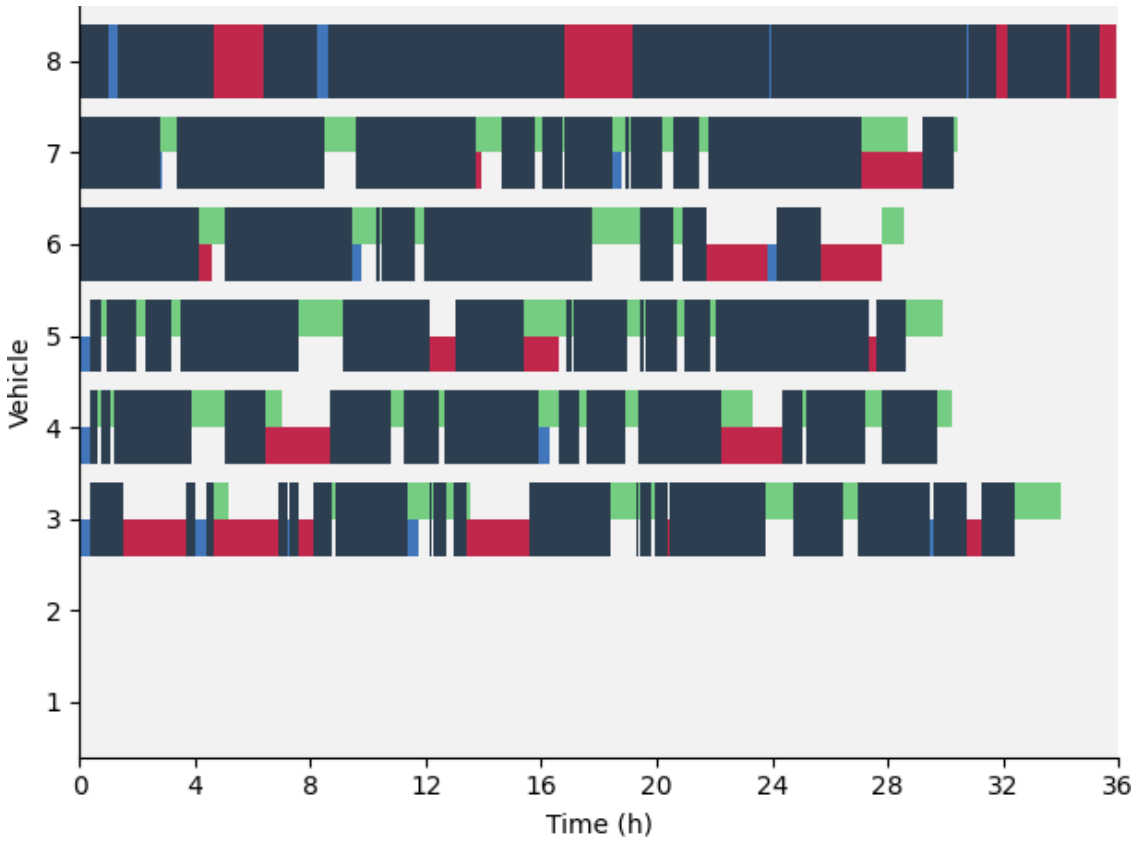

A Metaheuristic Algorithm for the Fleet Size and Mix Vehicle Routing Problem with Pickups and Deliveries

Developing an Approach for Finding Solutions to Realistic Large-Scale Instances

Master's Thesis in
Engineering Mathematics and Computational Science
& Computer Science - Algorithms, Languages and Logic

FATEMEH ALIBAKHSHI & PAWEL KASINSKI

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

A Metaheuristic Algorithm for the Fleet Size and Mix Vehicle Routing Problem with Pickups and Deliveries

Developing an Approach for Finding Solutions to Realistic Large-Scale Instances

FATEMEH ALIBAKHSHI
& PAWEL KASINSKI

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2025

A Metaheuristic Algorithm for the Fleet Size and Mix Vehicle Routing Problem with Pickups and Deliveries
Developing an Approach for Finding Solutions to Realistic Large-Scale Instances
FATEMEH ALIBAKHSHI & PAWEL KASINSKI

Supervisors: Sunney Fotedar & Toheed Ghandriz, Volvo Group Trucks Technology
Academic Supervisor: Jiaming Wu, Dept. of Architecture and Civil Engineering
Examiner: Balázs Adam Kulcsár, Dept. of Electrical Engineering

Master's Thesis 2025
Volvo Group Trucks Technology
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A visualization of a solution to the problem where a time series is shown for each used vehicle. The dark gray intervals indicate traveling, the blue and red intervals indicate loading and unloading of cargo respectively, and the green ones indicate charging. The vehicles 1 and 2 are part of the fleet but have not been used.

Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

A Metaheuristic Algorithm for the Fleet Size and Mix Vehicle Routing Problem with Pickups and Deliveries
Developing an Approach for Finding Solutions to Realistic Large-Scale Instances
FATEMEH ALIBAKHSHI & PAWEL KASINSKI
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis addresses an extended variant of the fleet size and mix vehicle routing problem with aspects such as split pickups and deliveries, different loading and unloading methods, and the charging of electric vehicles. The aim of the work has been to be able to efficiently solve large-scale instances of the problem based on real data. To tackle this objective, a metaheuristic algorithm has been developed based on the adaptive large neighborhood search framework. A procedure for destroying and repairing a solution has been designed and implemented such that the challenging aspects of the problem definition are handled appropriately. Moreover, the use of simulated annealing and the technique of applying noise to diversify the search have been explored. The algorithm was run on three realistic instances, in a set of shorter runs to investigate the effect of adding noise and using simulated annealing. Then, three relatively long runs were conducted to obtain definitive solutions to the instances. The algorithm managed to find a promising solutions to all three instances, and the objective value of the best solutions found were significantly lower than the objective values of the initial solutions.

Keywords: vehicle routing problem, pickup and delivery, heterogeneous fleet, split pickups and deliveries, charging, alns, adaptive large neighborhood search, heuristic, operations research, logistics

Acknowledgements

First and foremost, we would like to thank Sunney Fotedar. Thank you for showing great interest in the project, always finding time for discussions with us, and for supporting us from the beginning to the end. Your input has truly helped us shape this master thesis.

We would also like to thank Jiaming Wu; for sharing insights about his experience with the ALNS framework, and for giving us advice regarding both the project and how the master thesis should be written.

Furthermore, we would like to express our gratitude to Toheed Ghandriz for proposing this project and introducing us to it, and finally, we would like to thank Balázs Adam Kulcsár for keeping in touch with us throughout the project and for making the examination process feel straightforward.

Fatemeh Alibakhshi & Pawel Kasinski, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms used in this thesis, listed in alphabetical order.

Acronym	**Definition**
ALNS	Adaptive Large Neighborhood Search
AST	Additional Semitrailer
BEV	Battery Electric Vehicle
EV	Electric Vehicle
FSMVRP	Fleet Size and Mix Vehicle Routing Problem
GA	Genetic Algorithm
LNS	Large Neighborhood Search
MILP	Mixed-Integer Linear Programming
OBL	On-Board Lift
OBW	On-Board Waiting
PDPTW	Pickup and Delivery Problem with Time Windows
SA	Simulated Annealing
SC	Straddle Carrier
SoC	State of Charge
TS	Tabu Search
VNS	Variable Neighborhood Search
VRP	Vehicle Routing Problem

Contents

List of Figures

List of Tables

1

Introduction

Heavy-duty vehicles account for approximately two percent of vehicles on the road, but contribute to over 25 percent of greenhouse gas emissions from road transport in the European Union [1]. To reduce the climate footprint of this group of vehicles, a proposed solution is electrification. However, as electric heavy-duty vehicles are being commercialized, new challenges in infrastructure, routing, and investment planning emerge for customers of such vehicles. As customers want to make sure that their businesses will be profitable after investing in, for example, battery electric trucks, it is desirable to be able to estimate the economic effects that the purchase will have on their operations. Moreover, once a customer has extended or replaced their fleet of trucks with battery electric trucks, they want to plan their deliveries in an optimal way to minimize their costs while meeting the demands of their customers. To address these problems, it is necessary to be able to solve a new variant of the vehicle routing problem in an efficient manner.

1.1 The Vehicle Routing Problem

The vehicle routing problem (VRP) is an established problem, and determining an optimal solution is known to be NP-hard [2]. In general, the problem is to find routes originating from a depot for vehicles in a fleet such that goods are delivered to a set of customers while costs are kept to a minimum. Since the problem first appeared in an academic paper by George Dantzig and John Ramser in 1959, a multitude of variants have been introduced and studied [3], [4]. One of the variants is referred to as the fleet-size and mix-vehicle routing problem (FSMVRP). The differing aspects of FSMVRP compared to VRP is that the fleet size, i.e. the number of vehicles used, is not predetermined, and that the costs and capacities associated with the vehicles can vary [5]. In this thesis, a more complex vehicle routing problem that incorporates the aspects of FSMVRP is to be solved. Below, in Figure 1.1, a simple instance of the vehicle routing problem and an example solution to it are shown.

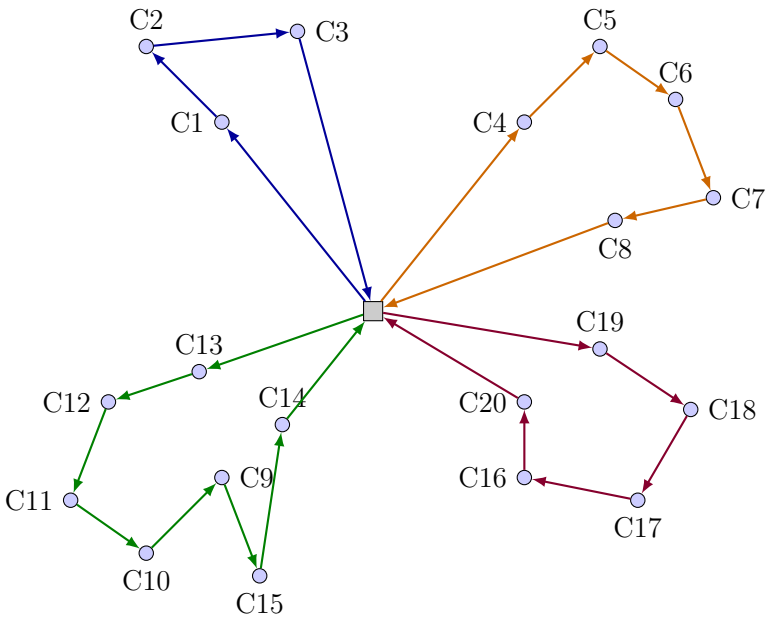

Figure 1.1: *An illustration of a solution with four vehicle routes to a vehicle routing problem instance with a central depot.*

1.2 Overview of the Problem

In the variant of the VRP that will be treated in this master thesis, the fleet of vehicles can include both diesel vehicles and electric vehicles. The vehicles can have different mass capacities, volume capacities, and in the case of electric vehicles, battery capacities. Similarly to FSMVRP, not all vehicles from the fleet have to be used in the solution. The addition of electric vehicles is significant as it comes with the added challenge of planning when and how to charge when searching for a solution.

An instance of the problem is represented as a graph where the locations are nodes and energy consumption values are arcs. Since the vehicles have varying specifications, they have differing energy values for the arcs. A node can be a depot, customer location, or a standalone charging station. At a customer node, a vehicle can either load or unload cargo, but a necessary condition is that the vehicle and the customer have matching loading types. While loading or unloading, an electric vehicle may also charge in parallel if possible at the considered customer. Furthermore, each vehicle start from a designated home depot location, and has to end its route at this location.

The problem is a pickup and delivery VRP variant, meaning that cargo is loaded at customer locations instead of at the depot and is then unloaded at other customer locations. A pair consisting of a pickup location and the corresponding delivery location will be denoted as a request, and each request will be associated with the amount of mass to be transported between these nodes. Moreover, since the mass

associated with a request can exceed the mass capacity of a vehicle, the request may need to be served by multiple vehicles carrying part of the cargo, or by the same vehicle going back and forth. This aspect of the problem will be referred to as split pickups and deliveries. At a later point in the thesis, in chapter 4, the problem is defined more rigorously using mathematical notation.

1.3 Context

Prior to this master thesis, extended versions of FSMVRP have been studied, and recently an approach using column generation was proposed for a variation that shares many similarities with the problem that will be dealt with in this thesis [6]. Column generation is an exact approach, and exact approaches are one of the main directions where research effort is put in the field. However, the vehicle routing problem cannot be solved by a polynomial-time algorithm and the time to find an optimal solution scales exponentially with regard to instance size. Thus, it is not feasible to find optimal solutions for large instances. Therefore, another prominent research direction in the field regards inexact approaches that utilize, for example, metaheuristics, matheuristics, or deep reinforcement learning [7], [8], [9]. These so called heuristic approaches have been shown to be effective for large-scale instances of vehicle routing problems and have performed well in international VRP competitions [10].

1.4 Aim

The aim of this master thesis is to study how a heuristic method can be applied to solve the previously described extended variant of FSMVRP. More specifically, the focus is to adapt the Adaptive Large Neighborhood Search (ALNS) framework to be able to find good solutions for relatively large instances of the variant of the problem, in a time efficient manner. As such, the research question for this thesis can be formulated as follows:

- How to develop an algorithm for solving the defined problem such that its complexities are handled and large-scale instances can be solved efficiently?

1.5 Delimitations

To reduce the scope of the project to some extent, a few assumptions have been made. First, it is assumed that the pickups and deliveries will always be one to one relations. In other words, if it is known where certain cargo has been picked up, it is known where it should be delivered.

Second, it has been assumed that a vehicle can only load or unload at a node, and never perform both actions.

Third, it is assumed that it is possible to serve all requests in the considered problem instances using the vehicles from the fleet. Thus, solutions where some customer are unable to be served will not be accepted and consequently there is no need for a request bank for storing such requests like the one proposed by Røpke and Pisinger in the academic paper "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows" [11].

2

Literature Review

The Vehicle Routing Problem (VRP) is one of the most studied problems in transportation and logistics. Over the years, researchers have developed many versions of this problem to reflect more realistic conditions, such as time windows, split pick-ups and deliveries, and heterogeneous vehicles. Recently, there has been growing interest in solving VRPs with electric vehicles (EVs), due to the increasing demand for more sustainable transportation, especially in urban areas.

2.1 Classical and Metaheuristic Approaches to the Vehicle Routing Problem

Traditional methods for solving VRPs include classical heuristics like the nearest neighbor algorithm and Clarke-Wright savings algorithm [12], [13]. These methods are fast and simple, but they often produce low quality solutions when the problem includes complex constraints.

To improve solution quality, many researchers turned to metaheuristics, such as Genetic Algorithms (GA) [14], Tabu Search (TS) [15], Simulated Annealing (SA) [16], and Variable Neighborhood Search (VNS) [17]. These methods are more flexible and better at exploring the solution space, especially in large and complex problems.

Among these, Adaptive Large Neighborhood Search (ALNS) has become one of the most popular algorithms for solving rich VRPs. It was introduced by Røpke and Pisinger [11] and improves on the original Large Neighborhood Search by using an adaptive mechanism to select the best operators during the search. ALNS has been successfully applied to many versions of VRP and is known for its flexibility and good performance.

2.2 Energy-Aware and Electric Vehicle Routing Problems

As more companies adopt electric vehicles, routing models must also include energy constraints, such as battery capacity and the need to stop at charging stations.

Several studies have explored how to include these aspects in VRP models.

For example, Hiermann et al. [18] studied a version of the problem where the fleet includes both electric and diesel vehicles. Their model considers battery limits and charging needs, while also respecting time windows and vehicle capacities. They used a hybrid approach that combines ALNS with an exact Set Partitioning model. This allows them to explore many possible routes using ALNS and then choose the best ones using optimization.

Keskin and Çatay [19] also used a hybrid method for the electric VRP with time windows. They focused on fast charging stations and used ALNS to generate routes and a mathematical model to fine-tune charging decisions. Their results show that combining metaheuristics with exact models can lead to better solutions.

Zhou et al. [20] added even more realism by including waiting times at charging stations. Their model uses a two-step approach: first, it finds possible routes using heuristics, and then it adjusts charging times using a queueing model. This is important because, in real life, EVs often have to wait at busy charging stations. Even short delays can affect delivery times and energy usage.

These studies all support the idea that routing and charging decisions should not be made separately. They show that hybrid models, especially those using ALNS, are well-suited for solving modern VRPs that involve energy constraints and complex delivery requirements.

2.3 Machine Learning for Vehicle Routing Problems

In recent years, machine learning (ML) has started to play a role in solving VRPs. ML can help by learning patterns from past data or by improving decision-making in real time [21], [22].

Some studies use supervised learning methods like decision trees or neural networks to predict travel time, energy use, or customer demand. For example, supervised models have been used to forecast customer order volumes, estimate travel durations under uncertainty, and predict vehicle energy consumption [23], [24]. These predictive models can support traditional algorithms or help reduce the search space, leading to more efficient solutions.

More advanced approaches use reinforcement learning (RL), where the model learns routing strategies by interacting with the environment. Deep reinforcement learning, including methods like Pointer Networks [25] and attention-based models trained via policy gradients [26], has been applied to VRP and related problems. These neural combinatorial optimization methods aim to build solutions directly, without the need for handcrafted heuristics, and have shown promising results in routing, scheduling,

and other discrete optimization tasks.

However, ML-based methods still face challenges. They often require large amounts of training data, can be difficult to generalize to new instances, and may be hard to scale to very large or complex problem settings [27]. Also, they are not yet widely used in practical logistics environments, partly due to their complexity and limited interpretability.

Although ML is not the main focus of this thesis, it is an important research direction and could be combined with metaheuristics in future work. For example, ML models could be used to guide the selection of operators in ALNS or to estimate energy use during routing.

2.4 Comparative Analysis of Solution Methods

Table 2.1 provides a summary of the most common approaches used to solve VRPs, comparing their main strengths and weaknesses. Each method has its own advantages, depending on the problem size, constraints, and computational resources. Classical heuristics are fast but often not suitable for rich VRPs. Metaheuristics offer better solution quality and are more flexible but need careful tuning. Exact methods guarantee optimality, but are limited to small instances due to high computational cost. ALNS strikes a balance between flexibility and performance, especially in large and complex problems, whereas machine learning-based models are promising but still developing and face challenges with generalization and data requirements.

Table 2.1: *Overview of common optimization techniques for VRPs*

Method Type	**Strengths**	**Weaknesses**
Classical Heuristics	Easy to implement, fast	Poor performance on complex or large-scale problems
Metaheuristics (e.g., ALNS)	Flexible, good solution quality, handles rich constraints	Requires parameter tuning, no optimality guarantee
Exact Methods	Optimal solutions, mathematically rigorous	High computation time, limited scalability
ML-based Models	Learns from data, adapts to real-time input	Requires large datasets, limited practical deployment

2.5 Research Gaps and Summary

To summarize, many different approaches have been used to solve VRPs, including classical heuristics, metaheuristics, exact methods, and machine learning. Among

these, ALNS stands out as an effective framework for solving complex, real-world problems. It offers a good balance between performance and flexibility, and it works well when combined with other methods.

There are still important challenges in this area. One of them is how to properly handle split pickups and deliveries. Some earlier approaches address this by modeling split loads explicitly, such as the work by Nowak et al. [28], which introduces a method for the Pickup and Delivery Problem with Split Loads. While this provides more flexibility than simple node duplication, it also adds significant modeling and computational complexity. There remains a need for heuristic methods that can handle split service in a more integrated and scalable way, especially in cases where requests may be shared across multiple vehicles or revisited within the same route.

Another major challenge is how to integrate electric vehicle charging into the routing process. In many cases, charging is planned ahead of time, but in reality, conditions can change, such as stations might be full, or vehicles might use more energy than expected. Dealing with this in real time, especially under uncertainty, is still very difficult.

In this work, we have focused on addressing both of these challenges. We investigate a more flexible way to handle split pickups and deliveries, and we also explore how charging decisions can be better integrated into routing. Both are important steps toward building more realistic and adaptable vehicle routing systems.

3

Theoretical Foundations

This chapter presents the theoretical foundations and algorithms underlying the solution approach used in this thesis. It begins by presenting the Vehicle Routing Problem (VRP) from a mathematical perspective, and explaining the computational complexity of the problem and its variants. This motivates the use of heuristic and metaheuristic methods, as exact algorithms become impractical for large or realistic instances. The core of the chapter, however, focuses on two powerful metaheuristics: Large Neighborhood Search (LNS) and its extension, Adaptive Large Neighborhood Search (ALNS). These techniques are particularly effective for solving complex and large-scale VRP instances that involve multiple constraints, such as capacity limits or heterogeneous fleets. LNS and ALNS form the backbone of the solution strategy adopted in this study.

3.1 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a well-known problem in transportation and logistics [29], [2] , [4]. It deals with planning the best routes for a fleet of vehicles that deliver goods from one central location (called a depot) to several customers. Each customer needs a specific amount of goods, and each vehicle can only carry a limited load. The goal is to reduce the total delivery cost while making sure that every customer is visited only once, no vehicle carries more than it should, and all the planned routes are possible to follow. A comprehensive treatment of this model and its variants can be found in Toth and Vigo [30].

Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a complete directed graph, where $\mathcal{V} = \{0, 1, 2, \ldots, n\}$ is the set of nodes, with node 0 denoting the depot and nodes 1 through n representing customers. Let $\mathcal{K} = \{1, 2, \ldots, m\}$ denote the set of vehicles. The set of arcs is defined as $\mathcal{A} = \{(i, j) \in \mathcal{V} \times \mathcal{V} \mid i \neq j\}$, where each arc (i, j) is associated with a cost $c_{ij} \geq 0$.

Each customer $i \in \mathcal{V} \setminus \{0\}$ has a demand $d_i > 0$, and each vehicle $k \in \mathcal{K}$ has a capacity limit Q.

The decision variables are defined as follows:

$u_i^k \in \mathbb{R}_{\geq 0}$ represents the cumulative load of vehicle k upon arrival at customer i.

$$x_{ij}^k = \begin{cases} 1, & \text{if vehicle } k \text{ travels directly from node } i \text{ to node } j, \\ 0, & \text{otherwise.} \end{cases}$$

$$y_i^k = \begin{cases} 1, & \text{if customer } i \text{ is visited by vehicle } k, \\ 0, & \text{otherwise.} \end{cases}$$

The VRP can then be formulated as:

$$\min \quad \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij}^k \tag{3.1}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{K}} y_i^k = 1 \qquad \forall i \in \mathcal{V} \setminus \{0\} \tag{3.2}$$

$$\sum_{j \in \mathcal{V} \setminus \{i\}} x_{ij}^k - \sum_{j \in \mathcal{V} \setminus \{i\}} x_{ji}^k = \begin{cases} 1, & \text{if } i = 0 \\ 0, & \text{if } i \in \mathcal{V} \setminus \{0\} \end{cases} \qquad \forall i \in \mathcal{V},\ k \in \mathcal{K} \tag{3.3}$$

$$y_i^k = \sum_{j \in \mathcal{V}} x_{ij}^k \qquad \forall i \in \mathcal{V} \setminus \{0\},\ k \in \mathcal{K} \tag{3.4}$$

$$u_i^k - u_j^k + Q x_{ij}^k \leq Q - d_j \qquad \forall (i,j) \in \mathcal{A},\ k \in \mathcal{K} \tag{3.5}$$

$$d_i \leq u_i^k \leq Q \qquad \forall i \in \mathcal{V} \setminus \{0\},\ k \in \mathcal{K} \tag{3.6}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A},\ k \in \mathcal{K} \tag{3.7}$$

$$y_i^k \in \{0,1\} \qquad \forall i \in \mathcal{V},\ k \in \mathcal{K} \tag{3.8}$$

Objective (3.1) minimizes the total cost incurred by all vehicle routes. Constraint (3.2) ensures that each customer is visited exactly once by a single vehicle. Flow conservation for each vehicle is maintained through constraint (3.3), which ensures that the number of routes entering and leaving each node is balanced. Constraint (3.4) links the routing variables to the customer assignment decisions. Subtour elimination and capacity feasibility are enforced by the Miller–Tucker–Zemlin (MTZ) constraints in (3.5), while Constraint (3.6) bounds the cumulative vehicle load. Finally, Constraints (3.7) and (3.8) enforce the binary nature of the routing and customer assignment variables, respectively.

This VRP formulation supports explicit modeling of individual vehicle routes and is widely used in extensions involving heterogeneous fleets, time windows, or electric vehicles [30].

3.1.1 The Complexity of the Vehicle Routing Problem

The Vehicle Routing Problem is known to be NP-hard [31], meaning that the computational effort required to solve it exactly increases exponentially with problem size. As a result, solving large instances optimally is often impractical, particularly in real-world contexts where timely decisions are crucial.

The computational complexity of the VRP has been the subject of extensive research. A seminal study by Lenstra and Rinnooy Kan [31] analyzed the complexity of various vehicle routing and scheduling problems, showing that many of them are computationally intractable when approached with exact methods, especially as the number of customers, constraints, or vehicle types grows.

Due to these challenges, most practical applications rely on heuristic or metaheuristic approaches. Although these methods do not guarantee optimality, they can produce high-quality solutions within acceptable computation times. As a result, they are widely used in real-world applications such as logistics, freight distribution, and transportation planning.

3.2 Large Neighborhood Search

Large Neighborhood Search (LNS) is a heuristic method designed to solve complex combinatorial problems with large solution spaces. It was first applied to vehicle routing by Shaw [32], and later extended and improved by Pisinger and Røpke [11], [33], who introduced adaptive mechanisms to enhance its performance.

The key idea in LNS is to change a large part of the current solution, instead of making small changes like traditional local search methods. This helps avoid getting stuck in a low-quality solution and gives the algorithm a better chance to find an improved one. The method works in two steps: it destroys part of the solution, and then repairs it to create a new one.

3.2.1 Fundamental Concepts

Consider a combinatorial optimization problem defined over the set of feasible solutions $\mathcal{X}$. The goal is to find a solution $x \in \mathcal{X}$ that minimizes a given cost function $c(x)$, formally expressed as:

$$\begin{aligned} \text{minimize} \quad & c(x) \\ \text{subject to} \quad & x \in \mathcal{X}, \end{aligned}$$

where $c(x)$ denotes the objective value (or cost) associated with solution x.

In LNS, we use two important steps:

> **Destroy step:** This removes some parts of the current solution, creating a partial or even broken solution.

Repair step: This tries to fix the broken solution and make it valid again.

Let's call the set of broken or incomplete solutions $\mathcal{X}^{\text{partial}}$. Then, we define:

Destruction operator $d : \mathcal{X} \to \mathcal{X}^{\text{partial}}$ selectively removes a subset of elements from a solution, thereby creating a partial structure.

Repair operator $r : \mathcal{X}^{\text{partial}} \to \mathcal{X}$ reconstructs a feasible solution by reinserting the removed elements.

$$x \xrightarrow{\text{destroy}} x^{\text{partial}} \xrightarrow{\text{repair}} x',$$

The neighborhood explored by LNS around a solution x is not defined directly, but instead through a process. It includes all the possible solutions that can be created by first using a destruction operator to partially break the current solution, and then applying a repair operator to rebuild it. This process can be described formally as:

$$\mathcal{N}(x) = \{r(d(x)) \mid d \text{ and } r \text{ are applied to } x, \text{ and } x \in \mathcal{X}\}.$$

Instead of listing all possible neighboring solutions, LNS selects a few candidates from this large neighborhood in each iteration. This makes it possible to explore the solution space efficiently without needing to examine every option.

3.2.2 LNS Algorithmic Framework

The LNS works by gradually improving solutions through repeated destruction and repair steps. During its execution, it keeps track of three main variables: x which is the current solution being worked on; x^b, the best solution found so far; and x^t, a temporary solution created in each iteration.

The structure of the overall LNS process, as depicted in Algorithm 1, is based on the methodology proposed by Pisinger and Røpke [33].

Algorithm 1 Large Neighborhood Search

Require: An initial feasible solution x
Ensure: Best solution x^b
1: $x^b \leftarrow x$
2: **repeat**
3: Generate a candidate solution: $x^t \leftarrow r(d(x))$
4: **if** accept(x^t, x) **then**
5: Update current solution: $x \leftarrow x^t$
6: **end if**
7: **if** $c(x^t) < c(x^b)$ **then**
8: Update best solution: $x^b \leftarrow x^t$
9: **end if**
10: **until** a stopping criterion is satisfied
11: **return** x^b

At each iteration, the destroy and repair operators are applied to the current solution x to produce a new candidate solution x^t. This candidate may replace the current solution depending on an acceptance criterion, and if it improves the best-known solution, x^b is updated. The algorithm terminates once a stopping condition, such as a maximum number of iterations or a time limit, is met.

An illustration of this process is shown in Figure 3.1, where a portion of the solution is removed and later reintegrated in a new configuration, demonstrating how LNS explores different neighborhoods in the search space.

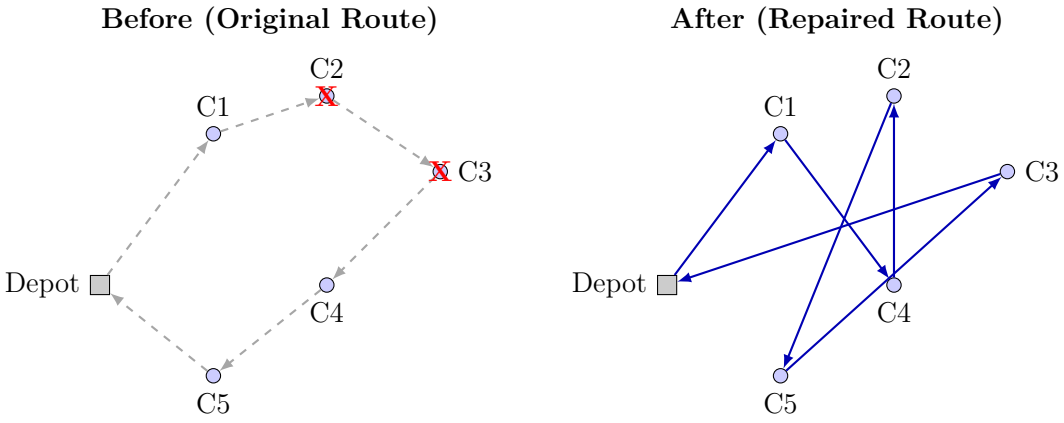

Figure 3.1: *Illustration of the LNS process.* ***Left:*** *The initial solution is shown as a gray dashed route visiting all customers. Two customers,* ***C2*** *and* ***C3****, are selected for removal during the* destruction phase. ***Right:*** *The repaired solution is constructed by reinserting the removed customers using a different route. The solid blue arrows show the new improved sequence, demonstrating the flexibility of LNS in exploring alternative neighborhoods.*

3.2.3 Acceptance Criterion

To balance exploitation (exploiting high-quality regions of the solution space) and diversification (exploring new areas), LNS often incorporates an acceptance criterion inspired by the principles of Simulated Annealing. As proposed by Pisinger and Røpke [33], the probability of accepting a candidate solution x^t, given the current solution x, is defined as:

$$\mathbb{P}\left(A(x^t, x)\right) = \begin{cases} 1, & \text{if } c(x^t) \leq c(x), \\ \exp\left(-\dfrac{c(x^t) - c(x)}{T}\right), & \text{otherwise,} \end{cases}$$

where $A(x^t, x)$ denotes the acceptance probability, $c()$ is the cost function, and $T > 0$ is a temperature parameter controlling the acceptance probability.

This acceptance criterion ensures that an improved solution ($c(x^t) \leq c(x)$) is always

accepted. However, if the candidate solution is worse, it may still be accepted with a probability that decreases as the cost difference increases and as the temperature T decreases. This allows the algorithm to escape local optima by accepting worse solutions.

At the beginning of the search, a high temperature encourages exploration by increasing the likelihood of accepting worse solutions. As the search progresses, the temperature is gradually reduced, shifting the focus toward exploitation. A common cooling schedule is given by:

$$T_{\text{new}} = \alpha \cdot T_{\text{old}},$$

where $0 < \alpha < 1$ is a cooling rate parameter that controls how quickly the temperature decreases over time.

3.2.4 Design of Destroy and Repair Operators

The performance of the LNS algorithm depends on how its destruction and repair operations are designed [33].

In the destroy phase, part of the current solution is removed to create a partial, and often infeasible solution [33]. This removal can be random to encourage exploration. Strategic destruction helps balance between exploring new solutions and improving existing ones.

The repair phase then rebuilds a feasible solution [33]. This can be done using fast greedy methods that reinsert elements with minimal negative impact, or through more advanced strategies like random insertion or based on optimization techniques, including mixed integer programming. These methods can lead to better-quality solutions, although they may require more computational effort.

Choosing how much of the solution to destroy is important: removing too little limits the search to nearby solutions, while removing too much can make the solution hard to repair, which increases computation time and might lower overall quality.

3.3 Adaptive Large Neighborhood Search

Adaptive Large Neighborhood Search (ALNS) is an extension of the traditional Large Neighborhood Search (LNS) method, first introduced by Røpke and Pisinger [11], [33]. Unlike standard LNS, ALNS does not rely on a single destroy and repair strategy. Instead, it chooses from several different heuristics during the search process. This flexibility makes the algorithm more robust and better able to adjust to different types of problems and stages in the search.

3.3.1 Motivation and Overview

Traditional LNS relies on a fixed pair of destroy and repair heuristics throughout the search process. In contrast, Adaptive Large Neighborhood Search (ALNS) extends this approach by employing a pool of heuristics and adjusting their selection probabilities based on performance feedback [33]. Instead of using the same pair of heuristics, ALNS evaluates the effectiveness of each one and updates their chances of being selected accordingly. This adaptive mechanism enhances the algorithm's ability to respond to varying problem characteristics.

From the perspective of neighborhood search, ALNS can be viewed as navigating among multiple large neighborhoods, each defined by a distinct destroy-repair heuristic pair. The choice of the next neighborhood is influenced by a selection strategy that favors heuristics with better performance. This adaptivity supports both diversification by exploring a broader portion of the solution space and exploitation by reinforcing the application of successful heuristics.

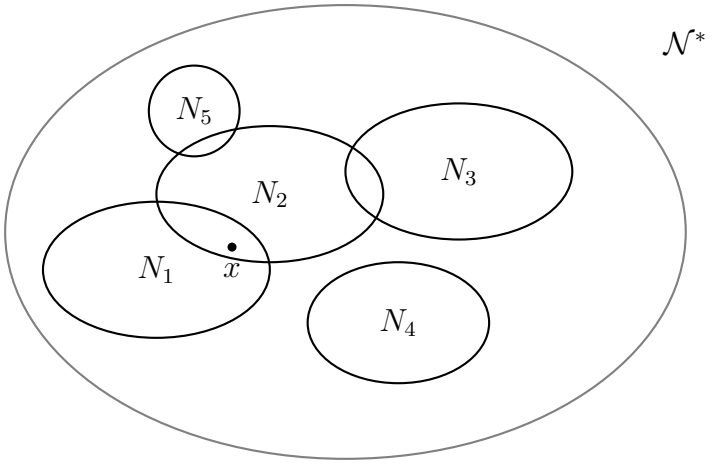

Figure 3.2: *Illustration of neighborhoods used by ALNS. The current solution is marked as x. ALNS operates on structurally diverse neighborhoods $N_1, \ldots, N_k$, each defined by a specific destroy-and-repair heuristic. These neighborhoods are all subsets of a larger neighborhood $\mathcal{N}^*$, defined by modifying up to q variables — the maximum degree of destruction allowed.*

3.3.2 ALNS Algorithmic Framework

Let:

$W^- = \{d_1, d_2, \ldots, d_k\}$ be the set of destroy heuristics,

$W^+ = \{r_1, r_2, \ldots, r_\ell\}$ be the set of repair heuristics.

Each operator $d_i \in W^-$ and $r_j \in W^+$ is associated with a weight ρ_i^- and ρ_j^+, respectively. These weights determine the selection probability of each heuristic, which is updated throughout the search based on performance feedback.

The overall procedure is summarized in Algorithm 2, following the structure proposed in [33].

Algorithm 2 Adaptive Large Neighborhood Search (ALNS)

Require: Initial feasible solution $x \in \mathcal{X}$
Ensure: Best solution found x^b
1: Initialize $x^b \leftarrow x$, $\rho_i^- \leftarrow 1$, $\rho_j^+ \leftarrow 1$ for all i, j
2: **repeat**
3: Select destroy $d_i \in W^-$ and repair $r_j \in W^+$ using roulette wheel selection based on ρ^-, ρ^+
4: Generate candidate: $x^t \leftarrow r_j(d_i(x))$
5: **if** accept(x^t, x) **then**
6: $x \leftarrow x^t$
7: **end if**
8: **if** $c(x^t) < c(x^b)$ **then**
9: $x^b \leftarrow x^t$
10: **end if**
11: Update weights ρ_i^-, ρ_j^+ according to reward function
12: **until** stopping criterion is satisfied
13: **return** x^b

Algorithm 2 begins with an initial feasible solution x, which is also recorded as the best-known solution x^b. For each destroy heuristic $d_i \in W^-$ and repair heuristic $r_j \in W^+$, an associated weight ρ_i^- or ρ_j^+ is initialized to 1. These weights are later used to guide the heuristic selection process.

During each iteration of the algorithm, one destroy and one repair heuristic are selected. The selected destroy heuristic d_i is applied to the current solution x, producing a partial solution. This partial solution is then completed using the selected repair heuristic r_j, resulting in a new candidate solution x^t.

The candidate solution x^t is accepted based on a predefined acceptance criterion that allows the algorithm to accept worse solutions in order to escape local optima and promote diversification. If x^t is accepted, it replaces the current solution x.

If the candidate solution is better than the best solution found so far ($c(x^t) < c(x^b)$), it is stored as the new best solution x^b. Regardless of acceptance, the weights ρ_i^- and ρ_j^+ of the selected heuristics are updated according to a reward mechanism, which increases the weight of heuristics that lead to better outcomes.

The algorithm continues until a stopping criterion is met, such as a time limit or a maximum number of iterations. At the end of the search, the best solution x^b encountered during the process is returned.

3.3.3 Destroy and Repair Operator Selection

ALNS selects destroy and repair heuristics at each iteration using an adaptive selection mechanism, as proposed by Pisinger and Ropke [33]. Each heuristic $d_i \in W^-$ (destroy) and $r_j \in W^+$ (repair) is assigned a weight, denoted by ρ_i^- and ρ_j^+, respectively. The probability of selecting a destroy heuristic d_i is computed by normalizing its weight against the sum of all destroy heuristic weights:

$$\phi_i^- = \frac{\rho_i^-}{\sum_{k=1}^{|W^-|} \rho_k^-}.$$

Similarly, repair heuristics are selected with probability:

$$\phi_j^+ = \frac{\rho_j^+}{\sum_{l=1}^{|W^+|} \rho_l^+}.$$

These probabilities form the basis for a roulette wheel selection process, where heuristics with higher weights are more likely to be chosen, but lower-weight heuristics still have a chance, allowing for a balance between exploitation and diversification.

After applying the selected pair of heuristics (d_a, r_b), the resulting candidate solution x^t is evaluated, and a reward score y is assigned to this heuristic pair based on the quality of x^t:

$$y = \begin{cases} w_1, & \text{if } x^t \text{ is a new global best,} \\ w_2, & \text{if } x^t \text{ improves the current solution,} \\ w_3, & \text{if } x^t \text{ is accepted,} \\ w_4, & \text{if } x^t \text{ is rejected,} \end{cases}$$

where the weights satisfy $w_1 > w_2 > w_3 > w_4 \geq 0$, reflecting the relative importance of each outcome.

The heuristic weights are then updated using an exponential smoothing update rule controlled by a parameter $\lambda \in [0, 1]$:

$$\rho_a^- \leftarrow \lambda \rho_a^- + (1 - \lambda) y, \quad \rho_b^+ \leftarrow \lambda \rho_b^+ + (1 - \lambda) y.$$

This update rewards heuristics that contribute to the search progress, while gradually discounting past performance, allowing the algorithm to adaptively learn which heuristics are most effective over time.

In summary, this adaptive mechanism, combining weighted roulette wheel selection based on performance scoring and updating, enables ALNS to adjust its search strategy to better explore and exploit the solution space.

3.3.4 Acceptance Criterion

Similar to the LNS approach, ALNS employs an acceptance criterion inspired by Simulated Annealing to decide whether to accept a candidate solution x^t. The acceptance probability is defined as:

$$\mathbb{P}\left(A(x^t, x)\right) = \begin{cases} 1, & c(x^t) \leq c(x), \\ \exp\left(-\dfrac{c(x^t) - c(x)}{T}\right), & \text{otherwise}, \end{cases}$$

where $A(x^t, x)$ denotes the acceptance probability, $c(\cdot)$ is the cost function, and $T > 0$ is a temperature parameter controlling the acceptance probability.

3.3.5 Design of Destroy and Repair Operators

The way destroy and repair operators are designed plays a major role in how well the ALNS algorithm performs. According to [33], different types of operators are used to balance exploration and improvement during the search.

Destroy operators that focus on diversification such as random removals methods, help the algorithm explore new parts of the solution space by breaking up the current solution structure. On the other hand, destroy operators focused on exploitation target specific elements, such as the most costly or critical components, to create opportunities for improved local solutions.

One popular destroy operator strategy removes elements that are similar in terms of node, timing, or type. This approach enables more focused and flexible exploration of local adjustments.

Repair operators can vary in how simple or complex they are. Greedy repair heuristics work quickly by adding elements back in places that cause the least extra cost. Regret-based methods, on the other hand, focus on reinserting the more difficult elements first, to avoid making worse decisions later. In some cases, it is even possible to use exact optimization techniques to solve small subproblems during the repair phase. However, since these methods can be time consuming, the algorithm may penalize slow repair strategies to maintain overall efficiency.

3.3.6 Properties and Strengths of ALNS

ALNS is a powerful and flexible framework for solving large and complex optimization problems. One of its key strengths is its ability to adapt during the search process. By learning from past performance, ALNS adjusts how it chooses heuristics, allowing it to balance between exploitation and diversification. This makes the method highly effective across different types of problem instances.

Another advantage of ALNS is that it requires relatively little parameter tuning. Most of the parameters are related to the adaptive selection process, not the specific

heuristics themselves. In addition, the framework is very flexible. new destroy and repair operators can easily be added, and the algorithm will automatically learn when to use them based on their performance.

Unlike methods such as Variable Neighborhood Search (VNS), which follow a fixed sequence of neighborhood structures, ALNS uses a probability-based approach that responds to how well different heuristics perform. This makes ALNS especially useful in problems where neighborhoods are defined by heuristics rather than strict mathematical rules.

Figure 3.3 illustrates the key difference between the traditional LNS and ALNS. In LNS, a fixed pair of destroy and repair heuristics is used throughout the search process. This approach can be effective but may limit flexibility across different problem landscapes.

ALNS extends this structure by maintaining a pool of heuristics and selecting a pair based on their historical performance. After each iteration, the algorithm updates the weights associated with each heuristic according to how well they performed, allowing it to gradually favor more effective strategies. This adaptive layer helps ALNS to better balance exploration and exploitation, improving its performance on a wide range of complex optimization problems [33].

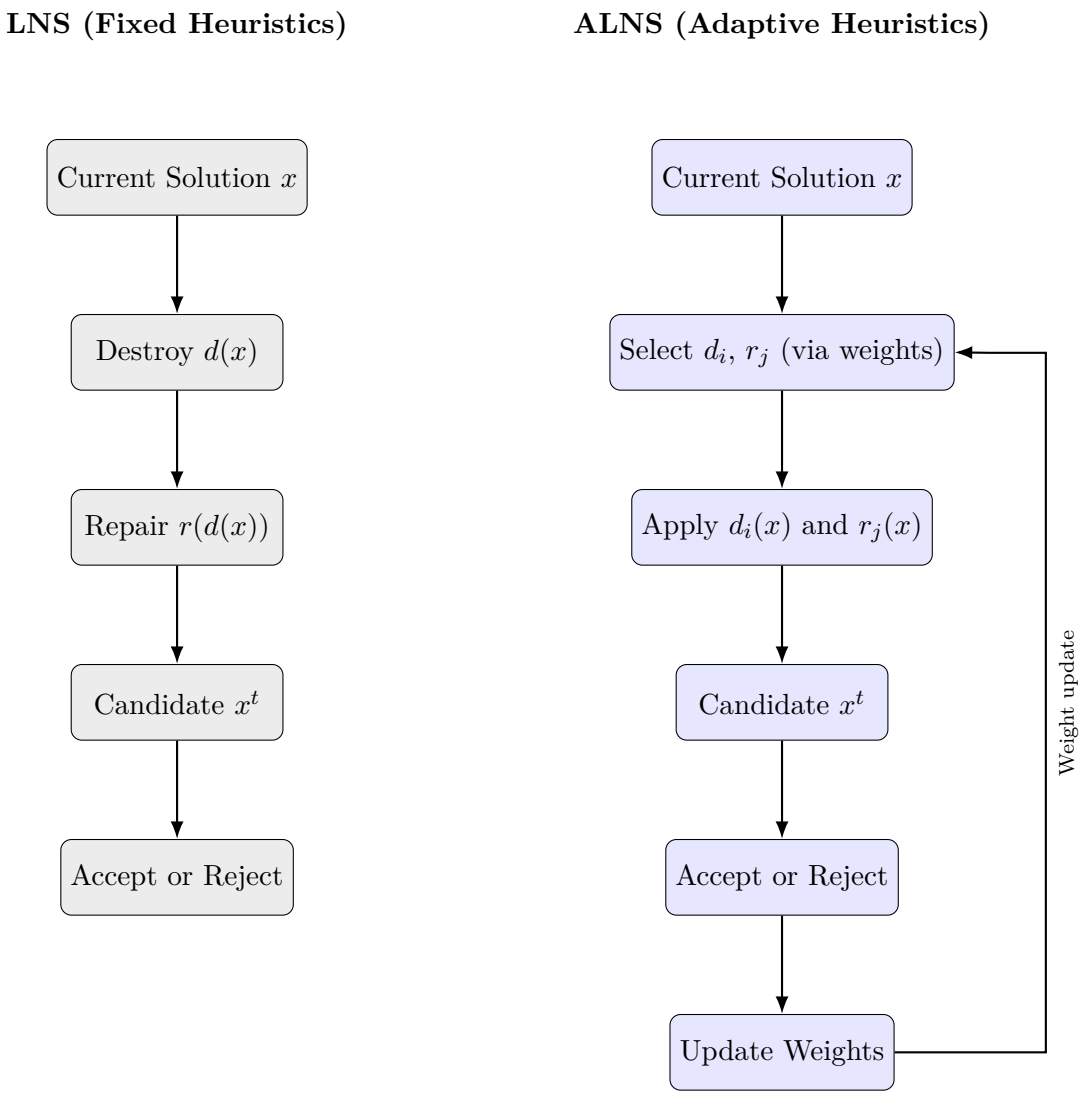

Figure 3.3: *Comparison of LNS and ALNS. LNS uses a fixed destroy/repair strategy, while ALNS adaptively selects from multiple heuristics and updates their weights based on performance.*

3.4 Dijkstra's Algorithm

Dijkstra's algorithm was invented in 1956 and is used for finding the shortest paths from one node, to all other nodes in a graph [34]. The algorithm starts at a given node and repeatedly visits the nearest unvisited node while keeping track of the shortest distance and the predecessor to each node.

In the algorithm, initial shortest distances are first set for all nodes. The value associated with the starting node is set to zero, while the values of the remaining nodes are initialized to infinity [34]. Then the following is done: the unvisited node with the lowest shortest distance is selected, and for each of its neighboring nodes, the distance to the neighboring node via the selected node is calculated by adding

the shortest distance to the selected node, and the distance between the selected node and the neighboring node. If this sum is lower than the saved shortest distance to the neighboring node, it becomes the new shortest distance to that node and the selected node is saved as the predecessor of the neighboring node. Finally, the currently selected nodes is marked as visited, and the procedure repeats, starting with a new unvisited node being selected.

Once all nodes have been visited, the shortest paths originating from the starting node can be extracted by starting at a certain node and going to the preceding node that was saved the shortest distance was found, then going to the preceding node of that preceding node, and so on until the starting node is reached [34]. At that point, a sequence of nodes has been obtained that makes up the shortest path.

4

Mathematical Model

This chapter presents the mathematical model developed to address the extended Fleet Size and Mix Vehicle Routing Problem (FSMVRP). The model is based on a flow formulation and integrates several real-world considerations, including heterogeneous vehicle types (electric and diesel), detailed loading and unloading operations, energy consumption, and the possibility for vehicles to revisit locations. The transportation network is represented as a directed graph, and the formulation incorporates practical constraints that reflect both logistical requirements and physical limitations commonly encountered in real-world operations.

4.1 Terminology

Before presenting the formal mathematical formulation, we introduce the key terms and concepts used throughout the model. These definitions provide the necessary context for understanding the structure of the transportation network, vehicle operations, and logistical constraints included in the problem at hand.

Node — A location in the network, representing either a customer node (for pickup or delivery) or a charging station node (for battery charging).

Arc — A directed connection between two nodes in the transportation network, representing a possible movement or traversal by a vehicle.

Fleet — The complete set of vehicles available for assignment, which may differ in volume capacity, mass capacity, or propulsion system (e.g., electric or diesel).

Home Depot — A central node in the transportation network representing the main facility. Each vehicle must complete its route at its corresponding home depot, although it may start from a different node.

Step A discrete index n representing a movement or transition in the route from one node to the next node, where $n = 1$ marks the first move.

Route A planned sequence of steps (nodes and arcs) followed by a vehicle from origin to destination.

Service Time The time spent at a node performing loading, unloading, or charging operations.

Loading/Unloading Methods
Cargo handling techniques available at nodes, with multiple types possibly supported per vehicle and location.

A more explanation of the loading and unloading methods considered in this study is provided below. These methods are based on practical handling processes described by Toheed Ghandriz in [35] and are incorporated with specific service time rates.

The model considers four types of loading and unloading operations:

(i) *Additional Semitrailer* (AST): Attaching or detaching an additional semitrailer to or from the vehicle.

(ii) *On-Board Lift* (OBL): Utilizing a lift mechanism mounted on the vehicle to handle loading or unloading.

(iii) *On-Board Waiting* (OBW): Waiting at the node while external lift vehicles perform cargo handling.

(iv) *Straddle Carrier* (SC): Using a straddle carrier at the node to transfer containers to or from the vehicle.

4.2 Problem Description

This section presents the description of the problem under consideration. The underlying structure is based on a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{A}$ is the set of directed arcs representing feasible movements between nodes.

The node set $\mathcal{V}$ is partitioned as:

$$\mathcal{V} = \mathcal{V}_0 \cup \mathcal{V}_c \cup \mathcal{V}_s,$$

where $\mathcal{V}_0$ includes the home depot, $\mathcal{V}_c$ represents customer locations, and $\mathcal{V}_s$ contains charging stations. Notably, a node may simultaneously belong to both $\mathcal{V}_c$ and $\mathcal{V}_s$ if it serves both functions.

The arc set $\mathcal{A}$ consists of all valid transitions between nodes:

$$\mathcal{A} = \{(i, j) \mid i, j \in \mathcal{V},\ i \neq j,\ i \neq 0_+,\ j \neq 0_-\}.$$

Each arc $(i, j) \in \mathcal{A}$ represents a direct route from node i to node j and, the home depot is modeled as two separate nodes: 0_- (the starting home depot) and 0_+ (the ending home depot).

4.3 Sets, Parameters, and Decision Variables

This section defines the fundamental components used in the mathematical formulation of the problem. The model relies on a set of indices and elements to represent the network.

4.3.1 Sets

The model is built using several sets that help define the structure of the problem. These sets are used as the basic indexing system throughout the mathematical model.

Table 4.1: *Sets used in the optimization model*

Set	Description
$\mathcal{K}_{\text{el}}$	Set of electric vehicles
$\mathcal{K}_d$	Set of diesel vehicles
$\mathcal{K}$	Set of all vehicles: $\mathcal{K} = \mathcal{K}_{\text{el}} \cup \mathcal{K}_d$
$\mathcal{V}_c$	Customer nodes
$\mathcal{V}_s$	Charging station nodes
$\mathcal{V}_o$	The home depot
$\mathcal{V}$	All nodes in the network: $\mathcal{V} = \mathcal{V}_s \cup \mathcal{V}_c \cup \mathcal{V}_o$
$\mathcal{V}_L$	Loading nodes
$\mathcal{V}_U$	Unloading nodes
$\mathcal{N}$	Set of steps: $\mathcal{N} = \{1, ..., N$[1]$\}$
$\mathcal{A}$	Set of arcs between nodes
$\mathcal{H}$	Loading and unloading methods

[1] N denotes the maximum number of steps across all vehicles, i.e., $N = \max_{k \in \mathcal{K}} N_k$, where N_k is the last step for vehicle k.

4.3.2 Decision Variables

The decision variables capture the routing behavior, product flows, energy consumption, and service activities of vehicles across the network. These are the key quantities optimized in the model.

Table 4.2: *Decision variables used in the optimization model*

Variable	Description
$y_{ij}^{k,n}$	Load carried by vehicle $k \in \mathcal{K}$ on arc $(i,j) \in \mathcal{A}$ at step $n \in \mathcal{N}$ [kg].
$m_{\text{Load},i}^{k,n,h}$	Quantity loaded on vehicle k using method $h \in \mathcal{H}$ at node $i \in \mathcal{V}_L$ during step n [kg].
$m_{\text{Unload},i}^{k,n,h}$	Quantity unloaded from vehicle k using method h at node $i \in \mathcal{V}_U$ during step n [kg].
$z_{ij}^{k,n}$	Energy consumed by vehicle k on arc (i,j) at step n [kWh].
$E_{ij}^{k,n}$	Energy needed for vehicle k to traverse arc (i,j) at step n [kWh].
$q_i^{k,n}$	Battery level of vehicle k upon arrival at node i at step n [kWh].
$p_i^{k,n}$	Amount of electricity charged by vehicle k at node i during step n [kWh].
$s_i^{k,n}$	Service time of vehicle k at node i during step n [h].
$\tau_i^{k,n}$	Arrival time at node i for vehicle k at step n [h].
β_k	Total working time (service time and travel time) of vehicle k [h].
α_k	Effective wage-incurring time of vehicle k [h].
$x_{ij}^{k,n}$	Binary variable indicating whether vehicle k travels on arc (i,j) at step n (1 if traveled, 0 otherwise).
δ_k^{fix}	Binary variable indicating whether vehicle k is used, i.e., performs at least one loading operation (1 if used, 0 otherwise).

4.3.3 Parameters

This subsection lists the model's fixed input data. These parameters define the operational environment and constraints under which optimization occurs.

Table 4.3: *Model parameters used in the optimization formulation*

Parameter	Description
c^d	Cost of diesel per liter (€/liter).
c_i^{el}	Electricity cost for charging at node $i \in \mathcal{V}$ [€/kWh].
c^w	Driver wage cost per hour [€/h].
c_k^f	Deployment cost associated with using vehicle $k \in \mathcal{K}$ [€].
ρ	Density of cargo [kg/m^3].
C_k^V	Maximum cargo volume capacity for vehicle k [m^3].
C_k^M	Maximum mass capacity for vehicle k [kg].
$D_{\text{Unload},i}$	Delivery demand at unloading node i [kg].
$D_{\text{Load},i}$	Pick-up demand of at loading node i [kg].
B^k	Battery capacity for electric vehicle k [kWh].
Γ_{ij}^k	Coefficient for calculating energy consumption per unit load on arc (i,j) for vehicle k [kWh/kg].[2]
Π_{ij}^k	Fixed energy consumption on arc (i,j) for vehicle k [kWh].[2]
t_{ij}	Travel time between node i and node j [h].
$t^{LU,h}$	Time per unit mass for loading/unloading using method h [h/kg].
$t_k^{\max}$	Maximum available working time for vehicle k [h].
r_g^{el}	Charging rate per electricity unit for charging type g [h/kWh].
M	A sufficiently large constant related to energy flow constraints [kWh].
Q	A sufficiently large constant used for constraint activation [kg].
Ω	Conversion constant for balancing diesel and electric energy units (liter/kWh).
$L_i^{k,h}$	Binary parameter indicating whether vehicle k is allowed to load or unload using method h at node i (1 if allowed, 0 otherwise).

[2]Both Γ_{ij}^k and Π_{ij}^k are derived from detailed vehicle dynamics and propulsion parameters not included in this report.

4.4 Objective Function of the Model

Using the sets, parameters, and decision variables introduced earlier, this section now presents the full mathematical model for the VRP, including all the necessary constraints.

$$\begin{aligned}\text{minimize} \quad & \underbrace{\sum_{k \in \mathcal{K}} c^w \cdot \alpha_k}_{\text{Wage cost}} \\ & + \underbrace{\sum_{k \in \mathcal{K}} \delta_k^{\text{fix}} \cdot c_k^f}_{\text{Vehicle deployment cost}} \\ & + \underbrace{\sum_{k \in \mathcal{K}_d} \sum_{n \in \mathcal{N}} \sum_{(i,j) \in \mathcal{A}} \Omega \cdot c^d \cdot z_{ij}^{k,n}}_{\text{Diesel energy cost}} \\ & + \underbrace{\sum_{k \in \mathcal{K}_{\text{el}}} \sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{V}_s} c_i^{\text{el}} \cdot p_i^{k,n}}_{\text{Electric charging cost}}\end{aligned} \tag{4.1}$$

The objective function (4.1) minimizes the total cost of routing a heterogeneous fleet consisting of both electric vehicles (EVs) and diesel vehicles. It integrates cost elements related to driver wages, fixed vehicle usage, fuel consumption, and electricity charging:

Wage cost: This term reflects the cost of active driver working time. It includes both service time and travel time, but is only counted if the vehicle performs at least one loading operation (i.e., when $\delta_k^{\text{fix}} = 1$). The variable α_k is activated through linear constraints mentioned in (4.5.4) that link it to the actual working time β_k and the binary service indicator δ_k^{fix}.

Vehicle deployment cost: Each vehicle that serves at least one customer (by performing a loading operation) is assigned a deployment cost. This cost is included in the model to encourage using fewer vehicles when possible and to avoid unnecessary vehicle usage.

Diesel fuel cost: This term models the energy cost of diesel vehicles based on energy consumption $z_{ij}^{k,n}$ over arc (i, j) at step n, scaled by a unit conversion factor Ω to match fuel and energy units. It is only applied to vehicles in the diesel subset $\mathcal{K}_d$.

Electricity charging cost: Electric vehicles incur a cost based on the amount of energy $p_i^{k,n}$ charged at node i, with cost rates c_i^{el}.

The structure ensures that costs are only applied when the vehicle is actually doing

something important; like loading products or using energy.

4.5 Model Constraints

This section introduces the mathematical constraints that guide how the vehicle routing problem works in this model. These rules make sure everything is feasible and realistic in terms of vehicle movements, product deliveries, energy use, and time limits. To make the explanation clearer, the constraints are organized into groups based on the main parts of the problem: routing, product flow, battery energy, time management, and how variables are defined.

4.5.1 Routing Constraints

The routing constraints control how vehicles move through the network, making sure the routes are logical and continuous. These rules ensure that each vehicle starts from the home depot, follows a step-by-step path through the network, and eventually arrives at its final destination.

$$\sum_{\substack{j\in\mathcal{V}:\\(j,i)\in\mathcal{A}}} x_{ji}^{k,n} - \sum_{\substack{j\in\mathcal{V}:\\(i,j)\in\mathcal{A}}} x_{ij}^{k,n+1} = 0, \qquad i \in \mathcal{V}\setminus\mathcal{V}_o, \quad k\in\mathcal{K}, \quad n\in\mathcal{N}\setminus\{N_k\} \tag{4.2}$$

$$\sum_{\substack{j\in\mathcal{V}:\\(0_-,j)\in\mathcal{A}}} x_{0_-j}^{k,1} = 1, \qquad k\in\mathcal{K} \tag{4.3}$$

$$\sum_{\substack{i\in\mathcal{V}:\\(i,0_+)\in\mathcal{A}}} \sum_{n\in\mathcal{N}} x_{i0_+}^{k,n} = 1, \qquad k\in\mathcal{K}. \tag{4.4}$$

Constraint (4.2) makes sure that the vehicle routes are continuous. For each vehicle k, and each step n, the number of incoming arcs to node i must equal the number of outgoing arcs, except at the home depot.

Constraint (4.3) ensures that every vehicle begins its route at the home depot by enforcing exactly one outgoing arc from that at the first step.

Constraint (4.4) guarantees that each vehicle terminates its route at the end depot 0_+, by requiring exactly one incoming arc into this node across all steps.

4.5.2 Loading and Unloading Constraints

This group of constraints governs the flow of multiple product types, ensuring that vehicle capacity limits are respected, and that loading and unloading activities meet customer demands while preserving consistency throughout the network.

$$y_{ij}^{k,n} \leq C_k^M x_{ij}^{k,n}, \qquad (i,j) \in \mathcal{A}, \quad k \in \mathcal{K}, \quad n \in \mathcal{N}, \tag{4.5}$$

$$\frac{y_{ij}^{k,n}}{\rho} \leq C_k^V x_{ij}^{k,n}, \qquad (i,j) \in \mathcal{A}, \quad k \in \mathcal{K}, \quad n \in \mathcal{N}, \tag{4.6}$$

$$\sum_{i \in \mathcal{V}_L} \sum_{n \in \mathcal{N}} \sum_{h \in \mathcal{H}} m_{\text{Load},i}^{k,n,h} \leq Q \cdot \delta_k^{\text{fix}}, \qquad k \in \mathcal{K}, \tag{4.7}$$

$$\sum_{n \in \mathcal{N}} \sum_{h \in \mathcal{H}} m_{\text{Load},i}^{k,n,h} \leq C_k^M, \qquad i \in \mathcal{V}_L, \quad k \in \mathcal{K}, \tag{4.8}$$

$$\sum_{n \in \mathcal{N}} \sum_{h \in \mathcal{H}} \frac{m_{\text{Load},i}^{k,n,h}}{\rho} \leq C_k^V, \qquad i \in \mathcal{V}_L, \quad k \in \mathcal{K}, \tag{4.9}$$

$$\sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}} \sum_{h \in \mathcal{H}} m_{\text{Load},i}^{k,n,h} = D_{\text{Load},i}, \qquad i \in \mathcal{V}_L, \tag{4.10}$$

$$\sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}} \sum_{h \in \mathcal{H}} m_{\text{Unload},i}^{k,n,h} = D_{\text{Unload},i}, \qquad i \in \mathcal{V}_U, \tag{4.11}$$

$$\sum_{\substack{j \in \mathcal{V} \\ (i,j) \in \mathcal{A}}} y_{ij}^{k,n+1} - \sum_{\substack{j \in \mathcal{V}: \\ (j,i) \in \mathcal{A}}} y_{ji}^{k,n} = \sum_{h \in \mathcal{H}} m_{\text{Load},i}^{k,n,h}, \qquad i \in \mathcal{V}_L,\ k \in \mathcal{K}, n \in \mathcal{N}, \tag{4.12}$$

$$\sum_{\substack{j \in \mathcal{V}: \\ (j,i) \in \mathcal{A}}} y_{ji}^{k,n} - \sum_{\substack{j \in \mathcal{V}: \\ (i,j) \in \mathcal{A}}} y_{ij}^{k,n+1} = \sum_{h \in \mathcal{H}} m_{\text{Unload},i}^{k,n,h}, \qquad i \in \mathcal{V}_U,\ k \in \mathcal{K}, n \in \mathcal{N}, \tag{4.13}$$

$$m_{\text{Load},i}^{k,n,h} \leq D_{\text{Load},i} \,.\, L_i^{k,h}, \qquad i \in \mathcal{V}_L,\ h \in \mathcal{H},\ k \in \mathcal{K},\ n \in \mathcal{N}, \tag{4.14}$$

$$m_{\text{Unload},i}^{k,n,h} \leq D_{\text{Unload},i} \,.\, L_i^{k,h}, \qquad i \in \mathcal{V}_U,\ h \in \mathcal{H},\ k \in \mathcal{K},\ n \in \mathcal{N}. \tag{4.15}$$

Constraints (4.5) and (4.6) ensure that the load carried by each vehicle respects its mass and volume capacities, respectively. They enforce that the total quantity transported does not exceed the vehicle's physical constraints on any arc.

Constraint (4.7) ensures that the binary indicator δ_k^{fix} is set to one if vehicle k performs at least one loading activity. In cases where no loading occurs, the summation on the left hand side evaluates to zero, and the minimization objective naturally favors $\delta_k^{\text{fix}} = 0$, thereby avoiding unnecessary fixed vehicle and wage costs. The constant Q is set as a sufficiently large upper bound representing the maximum load a vehicle could reasonably carry, ensuring that the constraint activates only when actual service is performed.

Constraints (4.8) and (4.9) ensure that at every loading node, the total quantity of

loaded cargo must not exceed the vehicle's physical carrying capabilities in terms of weight and space.

Constraints (4.10) and (4.11) guarantee that each product is properly picked up from home depot and fully delivered to its destination. These rules ensure that all product demands are completely met throughout the network.

Constraints (4.12) and (4.13) ensure flow balance at each node in the network. Specifically, constraint (4.12) applies to loading nodes ($\mathcal{V}_L$) and ensures that the difference between outgoing and incoming flows equals the quantity of goods loaded at the node. Conversely, constraint (4.13) applies to unloading nodes ($\mathcal{V}_U$) and ensures that the difference between incoming and outgoing flows matches the quantity of goods unloaded. Together, these constraints preserve the continuity of the transported load across the vehicle routes.

Finally, constraints (4.14) and (4.15) make sure that loading and unloading operations are allowed. They check that products are only handled where they are actually available and that each vehicle is permitted to load or unload product at certain nodes, based on $L_i^{k,h}$.

4.5.3 Battery Energy Constraints

These constraints control how battery energy is used and managed for electric vehicles along their routes. They make sure the energy levels stay consistent with how the vehicles move, respect the battery capacity limits, and correctly account for any recharging that happens during the trip.

$$q_i^{k,1} = B^k, \qquad k \in \mathcal{K}_{\text{el}}, \quad i \in \mathcal{V}_o \setminus 0_+, \tag{4.16}$$

$$E_{ij}^{k,n} = \Gamma_{ij}^k \cdot y_{ij}^{k,n} + \Pi_{ij}^k, \qquad (i,j) \in \mathcal{A}, \quad k \in \mathcal{K}, \quad n \in \mathcal{N}, \tag{4.17}$$

$$E_{ij}^{k,n} - M(1 - x_{ij}^{k,n}) \leq z_{ij}^{k,n}, \qquad (i,j) \in \mathcal{A}, \quad k \in \mathcal{K}, \quad n \in \mathcal{N}, \tag{4.18}$$

$$z_{ij}^{k,n+1} - B^k(1 - x_{ij}^{k,n+1}) \leq q_i^{k,n} + p_i^{k,n} - q_j^{k,n+1}, \quad (i,j) \in \mathcal{A}, \; k \in \mathcal{K}_{\text{el}}, \; n \in \mathcal{N} \setminus \{N_k\}, \tag{4.19}$$

$$q_i^{k,n} + p_i^{k,n} \leq B^k, \qquad i \in V_s \setminus 0_+, \quad k \in \mathcal{K}_{\text{el}}, \quad n \in \mathcal{N}, \tag{4.20}$$

$$q_{0_+}^{k,N_k} + p_{0_+}^{k,N_k} = B^k, \qquad k \in \mathcal{K}_{\text{el}}. \tag{4.21}$$

Constraint (4.16) initializes the battery level of each electric vehicle k to its full capacity B^k at the beginning of the route.

Constraint (4.17) computes the total energy requirement $E_{ij}^{k,n}$ to traverse arc (i,j) at

step n. This energy includes two parts: one that depends on how much the vehicle is carrying, and another fixed part, Π_{ij}^{k},

Constraint (4.18) ensures consistency between routing decisions and energy consumption. When arc (i, j) is used ($x_{ij}^{k,n} = 1$), the consumed energy $z_{ij}^{k,n}$ must equal or exceed the required energy $E_{ij}^{k,n}$. If the arc is not traversed ($x_{ij}^{k,n} = 0$), the constraint is relaxed via a big-M to avoid enforcing energy requirements on inactive arcs.

Constraint (4.19) ensures that the battery level at the next visited node j accounts for the previous battery level, any energy recharged at node i, and the energy consumed along arc (i, j). The formulation applies only to electric vehicles and excludes the final step N_k, since there's no next step after that point.

Constraint (4.20) ensures that the total energy in the battery, does not exceed the vehicle's battery capacity B^k. This constraint applies to all charging nodes except the end depot 0_+, which is treated separately.

Earlier formulation where recharging at 0_+ was disallowed, constraint (4.21) allows the vehicle to recharge at the depot before completing its operations.

4.5.4 Time Constraints

Charging Rate and Charging Time Definition

Let r^{el} denote the *charging rate* (in hours) required to supply 1 kWh, defined as:

$$r^{\text{el}} = \frac{1}{P},$$

where P is the charging power in kW.

The *charging time* at a node depends on the quantity of energy charged. Given this, the charging time incurred by vehicle k at node $i \in \mathcal{V}_s$ during step $n \in \mathcal{N}$ is:

$$s_{ch,i}^{k,n} = r^{\text{el}} \cdot p_i^{k,n},$$

where $p_i^{k,n}$ is the amount of energy (in kWh) charged.

Service Time Definition

At each visited node $i \in \mathcal{V}$, the *service time*, $s_i^{k,n}$, must account for both charging and product handling activities. Since, charging, loading, and unloading can happen at the same time, the service time is defined as the maximum of these three durations:

$$s_i^{k,n} = \max\left\{s_{ch,i}^{k,n},\ s_{L,i}^{k,n},\ s_{U,i}^{k,n}\right\}, \quad \forall i \in \mathcal{V},\ k \in \mathcal{K},\ n \in \mathcal{N},$$

with loading and unloading times given by:

$$s_{L,i}^{k,n} = \sum_{h \in \mathcal{H}} t^{LU,h} \cdot m_{\text{Load},i}^{k,n,h},$$

$$s_{U,i}^{k,n} = \sum_{h \in \mathcal{H}} t^{LU,h} \cdot m_{\text{Unload},i}^{k,n,h}.$$

Here, $t^{LU,h}$ is the time per unit (e.g., per kg) for loading/unloading using method $h \in \mathcal{H}$.

Therefore, time constraints are given by:

$$\sum_{i \in \mathcal{V}} \sum_{n \in \mathcal{N}} s_i^{k,n} + \sum_{(i,j) \in \mathcal{A}} \sum_{n \in \mathcal{N}} t_{ij} \cdot x_{ij}^{k,n} \leq t_k^{\max}, \qquad k \in \mathcal{K}, \tag{4.22}$$

$$\sum_{i \in \mathcal{V} \setminus \{0_+\}} \sum_{n \in \mathcal{N} \setminus \{N_k\}} s_i^{k,n} + \sum_{(i,j) \in \mathcal{A}} \sum_{n \in \mathcal{N}} t_{ij} \cdot x_{ij}^{k,n} \leq \beta_k, \quad k \in \mathcal{K}, \tag{4.23}$$

$$\beta_k - W(1 - \delta_k^{fix}) \leq \alpha_k, \qquad k \in \mathcal{K}, \tag{4.24}$$

$$s_i^{k,n} + t_{ij} - t^{\max}(1 - x_{ij}^{k,n+1}) \leq \tau_j^{k,n+1} - \tau_i^{k,n}, \qquad (i,j) \in \mathcal{A},\ k \in \mathcal{K},\ n \in \mathcal{N} \setminus \{N_k\}, \tag{4.25}$$

$$\tau_{0_+}^{k,N_k} + s_{0_+}^{k,N_k} \leq t^{\max}, \qquad k \in \mathcal{K}, \tag{4.26}$$

$$\alpha_k \leq t_k^{\max} \cdot \delta_k^{\text{fix}}, \qquad k \in \mathcal{K}, \tag{4.27}$$

$$s_{ch,i}^{k,n} \leq s_i^{k,n}, \qquad i \in \mathcal{V}_s,\ k \in \mathcal{K}_{\text{el}},\ n \in \mathcal{N} \tag{4.28}$$

$$s_{L,i}^{k,n} \leq s_i^{k,n}, \qquad i \in \mathcal{V}_L,\ k \in \mathcal{K},\ n \in \mathcal{N} \tag{4.29}$$

$$s_{U,i}^{k,n} \leq s_i^{k,n}, \qquad i \in \mathcal{V}_U,\ k \in \mathcal{K},\ n \in \mathcal{N} \tag{4.30}$$

Constraints (4.22) represent the total working time of each vehicle k, composed of both the service time $s_i^{k,n}$ at visited nodes and the travel time t_{ij} over traversed arcs, and ensures that this total does not exceed the vehicle's allowable working duration $t_k^{\max}$, thus maintaining route feasibility with respect to legal or contractual working limits.

In contrast, constraint (4.23) introduces an auxiliary variable β_k that upper-bounds the same working time expression, again excluding activities at the end node 0_+ in order to decouple time feasibility from cost modeling. This is necessary because the final service at 0_+ typically represents a terminal recharge operation, which does not contribute to wage cost under the operational assumptions of this model.

Therefore, any service time at the end depot is deliberately excluded from wage cost calculations while still being allowed in the time feasibility constraint.

Constraint (4.24) links the working time associated with β_k to another variable α_k, that should incur wage costs. To control whether this cost is applied, the binary variable δ_k^{fix} is used. If the vehicle is active ($\delta_k^{\text{fix}} = 1$), the constraint forces α_k to be at least as large as β_k, meaning all of the working time is counted for wage purposes. If the vehicle is not used ($\delta_k^{\text{fix}} = 0$), the constraint becomes:

$$\beta_k - W \leq \alpha_k,$$

where W is a large constant that serves as a "Big-M" parameter in mixed-integer programming. Mathematically, this constraint has the following logic:

- If $\delta_k^{\text{fix}} = 1$, then the constraint becomes:

  $$\beta_k \leq \alpha_k,$$

  ensuring that the wage-incurring time α_k is at least equal to the actual working time β_k.

- If $\delta_k^{\text{fix}} = 0$, then the constraint becomes:

  $$\beta_k - W \leq \alpha_k.$$

  Since W is chosen such that $W \geq \max_k t_k^{\max}$, the left hand side becomes a large negative number (i.e., $\beta_k - W \ll 0$). Therefore, the inequality is always satisfied for any value of α_k.

Thus, W enables the conditional enforcement of wage cost: α_k must capture working time only if the vehicle is used. To ensure correctness, W should be chosen as:

$$W \geq \max_{k \in \mathcal{K}} t_k^{\max},$$

so that the logic holds in all cases.

Constraint (4.25) guarantees that the arrival time at the subsequent node reflects both the service time at the current node and the travel time required to reach the next node. Additionally, the term involving $t^{\max}(1 - x_{ij}^{k,n+1})$ acts as a big-M style that effectively deactivates the constraint when arc (i, j) is not traversed. This formulation is also instrumental in preventing subtours, as it links the sequence of time steps to the selected arcs, ensuring a coherent progression of time across the route.

The constraint (4.26) ensures that the total duration of the vehicle's route, from initial departure to the return to the depot, does not exceed the maximum allowed operational time.

Constraint (4.27) define another auxiliary variable α_k. The logic is as follows:

- If the vehicle performs at least one loading operation ($\delta_k^{\text{fix}} = 1$), then $\alpha_k = \beta_k$, meaning all working time contributes to wage cost.

- If the vehicle does not perform any customer service ($\delta_k^{\text{fix}} = 0$), then $\alpha_k = 0$, and no wage cost is accrued in the objective function.

This approach makes sure that wage costs are only applied to vehicles that actually perform deliveries or pickups. Vehicles that are only used for repositioning or charging do not add to the wage cost.

Finally, constraints (4.28)–(4.30) ensure that the charging time $s_{ch,i}^{k,n}$, loading time $s_{L,i}^{k,n}$, and unloading time $s_{U,i}^{k,n}$ at any visited node are each bounded by the service time $s_i^{k,n}$.

4.5.5 Non-Negativity and Binary Constraints

To keep the optimization model consistent and feasible, certain rules are applied to the decision variables. All continuous variables must have non-negative values, and all variables that represent logical choices must be binary.

$$y_{ij}^{k,n} \geq 0, \quad \forall (i,j) \in \mathcal{A},\ k \in \mathcal{K},\ n \in \mathcal{N},$$

$$E_{ij}^{k,n} \geq 0, \quad \forall (i,j) \in \mathcal{A},\ k \in \mathcal{K},\ n \in \mathcal{N},$$

$$z_{ij}^{k,n} \geq 0, \quad \forall (i,j) \in \mathcal{A},\ k \in \mathcal{K},\ n \in \mathcal{N},$$

$$m_{\text{Load},i}^{k,n,h} \geq 0, \quad \forall i \in \mathcal{V}_L,\ k \in \mathcal{K},\ n \in \mathcal{N},\ h \in \mathcal{H},$$

$$m_{\text{Unload},i}^{k,n,h} \geq 0, \quad \forall i \in \mathcal{V}_U,\ k \in \mathcal{K},\ n \in \mathcal{N},\ h \in \mathcal{H},$$

$$q_i^{k,n} \geq 0, \quad \forall i \in \mathcal{V},\ k \in \mathcal{K},\ n \in \mathcal{N},$$

$$p_i^{k,n} \geq 0, \quad \forall i \in \mathcal{V},\ k \in \mathcal{K},\ n \in \mathcal{N},\ g \in \mathcal{G},$$

$$s_i^{k,n} \geq 0, \quad \forall i \in \mathcal{V},\ k \in \mathcal{K},\ n \in \mathcal{N},$$

$$\tau_i^{k,n} \geq 0, \quad \forall i \in \mathcal{V},\ k \in \mathcal{K},\ n \in \mathcal{N},$$

$$\alpha_k \geq 0, \quad \forall k \in \mathcal{K},$$

$$\beta_k \geq 0, \quad \forall k \in \mathcal{K},$$

$$x_{ij}^{k,n} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{A},\ k \in \mathcal{K},\ n \in \mathcal{N},$$

$$\delta_k^{\text{fix}} \in \{0,1\}, \quad \forall k \in \mathcal{K}.$$

5

Implementation

To solve the problem at hand within a reasonable amount of time, the ALNS framework was adapted and implemented. The implemented algorithm uses the same objective function and constraints as presented in section 4, but is not related to the concept of linear programming. The basic idea of the ALNS framework is that many iterations are performed and in each iteration the currently accepted solution is destroyed and repaired in an attempt to find a better solution. For this problem, destruction is done by removing visits where certain requests are served from the current vehicle routes, and repairing is done by inserting visits so that the requests are once more served. There are multiple ways in which the visits associated with the requests can be removed and inserted, and in each iteration a removal method and an insertion method is chosen randomly. As the search progresses, the methods that contribute to new best solutions being found are rewarded and the probabilities of these methods being chosen in future iterations increase - this is what adaptive refers to in ALNS.

5.1 Rationale for Choosing the ALNS Framework

Among the many algorithms available for solving vehicle routing problems, ALNS was selected for this thesis because it offers a strong balance between performance, flexibility, and practicality. One of the key reasons for choosing ALNS is its ability to adapt during the search process, which is especially valuable for complex problems that include many different types of constraints.

The problem definition in this study is not a simple one, as it includes electric and diesel vehicles, split pickups and deliveries, mass and volume constraints, as well as charging decisions. Most traditional heuristics or metaheuristics struggle to handle this level of detail. In contrast, ALNS allows the user to define and combine custom destroy and repair operators, making it possible to guide the search in ways that respect the specific rules of the problem. As the algorithm runs, it keeps track of which operators are working best and adjusts their usage. This adaptive feature helps the algorithm focus on the most promising parts of the solution space without getting stuck too early.

Further, previous research supports the use of ALNS in energy-aware VRPs, as shown in the work by Røpke and Pisinger [11], Hiermann et al. [18], and Keskin and Çatay [19].

5.2 Solution Representation

Due to aspects of the problem such as split pickups and deliveries, and charging, a solution cannot simply consist of routes that are represented as sequences of nodes like in many other VRP variations. In this thesis, a solution is a collection of vehicle routes such that each vehicle has a route, and each route is a sequence of visits. A visit contains information about which node was visited, the amount of mass loaded/unloaded, and the amount of time that the vehicle stayed at the node. Additionally, it holds information about how much energy was charged during the visit. The visit objects contain relative values, e.g. mass change instead of total mass, which was a deliberate choice as it removes the need for updating the data in the objects after removals and insertions of visits are done. It is worth noting that some vehicles may not be used, but the solution will still contain routes for these vehicles. The reason for this is that a vehicle has to at least travel from the start depot node to the end depot node, which makes up a route. These two nodes have been created artificially and share the same geographical location, the location of the home depot in the instance.

In addition to the solution, i.e. the vehicle routes, requests objects are used to keep track of which vehicles have served which requests. A request contains information about the pickup node and the delivery node for the request, the total amount of cargo mass that needs to be transported to serve the request, how much cargo mass is left to serve, and which vehicles have served the request. From here on out, when referring to the removal or insertion of requests, what is meant is that visits in which the requests are served are removed or inserted from the routes in the solution.

To give a clearer idea of what the solution representation looks like, a simple example is presented below, in Table 5.1.

Vehicle ID	Node	Load Change (kg)	Loading Time (s)	Energy Charged (kWh)	Charging Power (kW)
1	1	0	0	0	-
1	2	+25 000	9000	100	250
1	3	-25 000	9000	0	-
1	4	0	0	200	250
2	1	0	0	0	-
2	2	+5000	1800	100	250
2	3	-5000	1800	0	-
2	4	0	0	200	250

Request ID	Pickup Node	Delivery Node	Initial Mass (kg)	Remaining Mass (kg)	Served By
1	2	3	30 000	0	1, 2

Table 5.1: *An example of a solution in the form of a table where every row is a visit, together with a table with the request that is served in the solution. In this case, there is a split pickup and delivery as one request is served by two vehicles.*

5.3 Removal Methods

In this subsection, the three removal methods used in this thesis are introduced. The random removal and worst removal methods have previously been presented in [11], and single route removal is based on the route elimination method from [20].

Each removal method selects which requests to remove using a distinct strategy, and once the requests have been selected, the visits in which they are served are removed from the solution. Moreover, visits done strictly to charge are removed, and charging at customer nodes is removed. This is done to avoid the situation where customer visits are removed from a route, but the charging stops coupled with those visits remain even though they have become unsuitable. When a request is removed, it is always removed from all the routes in which it is served. For instance, if the request would be removed from the solution shown in table 5.1, it would be removed completely, i.e. the visits to nodes two and three would be removed from the route of vehicle one and the route of vehicle two.

5.3.1 Random Removal

Given a removal percentage, the number of requests to remove is obtained by multiplying the removal percentage with the total number of requests and rounding. Then, this many requests are selected randomly and visits where these requests are

served are removed from the solution.

5.3.2 Worst Removal

Using this method, the requests that are placed in the worst positions are removed. Given a removal percentage, the number of requests to remove is decided. Subsequently, each request is removed from the current solution separately and the difference between the initial objective value and the new objective value is calculated. In each such iteration, the request is saved along with the calculated difference. Finally, the requests with the highest difference are removed. To reduce the risk of the algorithm removing the same requests each time the method is applied to a certain solution, there is a stochastic aspect when doing the removal: with some probability, that can be adjusted using the parameter *worst removal determinism rate*, the currently worst positioned request will be kept and the next in order will be considered for removal instead.

5.3.3 Single Route Removal

In this method, a random vehicle that contributes to serving the requests in the current solution is selected. Then, each request that is partially served by this vehicle is removed from the solution. This means that every request from a single vehicle route is removed, and that the routes of other vehicles that partially served the same requests are affected. In theory, this method can reduce the number of used vehicles as it may be possible to reinsert the removed requests into other vehicle routes, and thereby reduce the objective value by avoiding some deployment cost.

5.4 Insertion Methods

After requests have been removed during an iteration, they are reinserted using one of the insertion methods. The insertions methods include greedy insertion, and two types of regret insertions which have previously been used in [11], and restricted greedy insertions, a method introduced in this thesis.

Inserting a request means that a visit where a product is picked up, and a visit where this product is delivered are inserted into a route such that the delivery happens after the pickup. Each insertion method compares the best insertion options for the different requests, and then the request insertion that is the best overall is selected. This is done repeatedly until all requests have been inserted. The aforementioned procedure is explained using pseudocode in Listing 5.1 and Listing 5.2. It is worth noting that the different insertion methods determine which insertion is the best overall in different manners.

To determine the best insertion option for each request that needs to be inserted, a certain set of steps is executed (described using pseudocode in Listing 5.3). In this procedure, every option of adding a pickup visit and delivery visit for a specific request into the current solution is checked. When considering the option of

inserting pickup and delivery visits at specific positions in a route, the maximum amount of mass that can be served is calculated so that the mass or volume capacity of the vehicle is not exceeded at any point in the route. Then, the available loading and unloading types of the pickup and delivery nodes are compared with the loading/unloading types of the currently considered vehicle, and the shortest loading and unloading times are determined and selected. Following this, it is checked if the route created after the insertion would be feasible. If the route would not be feasible due to an electric vehicle not having enough energy to reach a node, there is an attempt to repair the route by adding charging, and then feasibility is checked again. If the route would not be feasible for any other reason or if it cannot be repaired, the insertion option is disregarded and the next option is tried.

After trying the insertion options for a specific request, feasible insertion options have been found if they exist. Out of these options, the one where the value obtained by dividing the increase in objective value by the mass carried is the lowest is saved. The division is done to avoid unfairly favoring insertions where less cargo mass is transported. The previously described steps are repeated for all the requests that need to be inserted (as shown in Listing 5.2), and then the insertion options are compared to determine which insertion should be committed to (which is done differently depending on the insertion method). Following this, the solution is modified by committing to the the overall best request insertion, and the cargo mass left to serve for the selected request is updated. Next, the procedure is done again, and this continues until all requests that were removed in this iteration are once again fully served (as shown in Listing 5.1), or until the algorithm cannot proceed due to there not being any feasible insertion options.

Below, the complete insertion process is described using pseudocode:

```
insertions(solution, requests_to_insert):
  while requests_to_insert is not empty:
    single_insertion(solution, requests_to_insert)

    if the request associated with the insertion is fully served:
      remove the request from requests_to_insert
```

Listing 5.1: *Pseudocode for the entry point of the procedure of inserting all removed requests.*

```
single_insertion(solution, requests_to_insert):
  overall_best_insertion = None

  for request in requests_to_insert:
    insertion = get_best_request_insertion(solution, request)

    if insertion better than overall_best_insertion:
      overall_best_insertion = insertion

  if there is no overall_best_insertion:
    raise exception
  else:
    commit to overall_best_insertion and update the solution
```

```
14      update the request served in overall_best_insertion
```

Listing 5.2: *Pseudocode for a function that selects the overall best feasible insertion, commits to it, and updates the solution.*

```
get_best_request_insertion(solution, request):
  feasible_insertions = []

  for vehicle, route in solution:
    for i from 1 to route.length():
      for j from i + 1 to route.length() + 1:
        new_route = copy of route
        mass_change = compute the max amount of mass that can be
                      carried between position i and j in the route
        loading_time = compute loading time based on the loading
                       options of the vehicle and the pickup node
        unloading_time = compute unloading time based on the loading
                         options of the vehicle and the delivery node

        pickup_visit = create visit using the pickup node of
                       request, mass_change, and loading_time
        delivery_visit = create visit using the delivery node of
                         request, -mass_change, and unloading_time

        insert pickup_visit into new_route so that it is placed before
        the visit currently at index i

        insert delivery_visit into new_route so that it is placed
        before the visit currently at index j

        check the feasibility of new_route
        if new_route not feasible due to violated energy constraints:
          try repairing the route by adding or adjusting charging
          check the feasibility of new_route
        if new_route not feasible:
          continue by checking the insertion for the next i and j

        route_cost_delta = new_route.cost() - route.cost()

        add information about the current insertion to
        feasible_insertions, including route_cost_delta / mass_change

  if feasible_insertions is empty:
    raise exception
  else:
    sort feasible_insertions ascendingly by route_cost_delta /
    mass_change

    best_insertion = the first item in feasible_insertions
    best_insertion.regret_value = compute the regret value using
                                  feasible_insertions

    return best_insertion
```

Listing 5.3: *Pseudocode for a helper function that returns the best insertion option for a specific request, out of the feasible insertion options that were found.*

5.4.1 Greedy Insertion

When greedy insertion is used, the feasible insertion with the lowest adjusted route cost delta is selected as the best insertion during the procedure. In other words, the insertion where the increase in objective value divided by the mass served in that insertion is the lowest. Dividing by the mass served in the insertion is significant because the mass carried affects the total weight of the vehicle, and thereby its energy consumption. If the route cost delta was not adjusted, there could be a bias in favor of insertions where less mass is served as these insertions would increase the energy usage less, so the route cost delta would be lower. This behaviour is unwanted as it could lead to more frequent transportations of small amounts of cargo, instead of fewer transportations with greater amounts of cargo.

The adjusted route cost delta for a feasible insertion i of request r can be defined mathematically as follows: $\Delta_{r,i} = \frac{c_{r,i} - c}{m_{r,i}}$ where $c_{r,i}$ is the route cost that would be obtained after performing insertion i of request r, c is the route cost prior to insertion, and $m_{r,i}$ is the mass transported between the pickup node and delivery node of request r in insertion i. The values $\Delta_{r,i}$ that exist for every feasible insertion i of a request r can be ordered from lowest to highest. Let this ordering of the values be denoted as $\Delta_r^1, \Delta_r^2, \Delta_r^3, ..., \Delta_r^k$. With such notation, greedy insertion can be defined as a method where the insertion i with the lowest $\Delta_{r,i}$ is committed to for the request r with the lowest Δ_r^1.

5.4.2 Regret-2 Insertion

When regret-2 insertion is used, the difference between the adjusted route cost deltas of the two cheapest feasible insertions is calculated for each request, i.e $v_r = \Delta_r^2 - \Delta_r^1$. The resulting value for each request r is denoted as its regret value, v_r. The request insertion with the highest regret value is considered the best. If the regret values of two different request insertions are the same, the insertion with the lowest Δ_r^1 is chosen, like in greedy insertion, as a tie breaker. Thus, regret-2 insertion can be defined as a method where the insertion i with the lowest $\Delta_{r,i}$ is committed to for the request r with the highest v_r value.

5.4.3 Regret-3 Insertion

Regret-3 is an extension of the regret-2 insertion method. The difference is that the regret value v_r for each request r is calculated in the following way instead: $v_r = \Delta_r^2 - \Delta_r^1 + \Delta_r^3 - \Delta_r^1$. When this method is used, the best request insertion is the one with the highest v_r, and in the case of a tie the greedy insertion criteria apply similarly as in the regret-2 insertion method. Formally, regret-3 insertion can be defined as a method where the insertion i with the lowest $\Delta_{r,i}$ is committed to for the request r with the highest v_r value. It should be noted that the regret value for a request can only be calculated in this method if there exist at least three feasible insertion options for that request. Compared to greedy insertion, the advantage of regret-2 and regret-3 insertion is that these methods avoid deferring expensive

insertions.

5.4.4 Restricted Greedy Insertions

This method is different from the others as it does not consider all requests when determining the best insertion. Instead, the requests to insert are shuffled, and each request is then inserted greedily in order. This approach is nondeterministic and more time efficient as it considers fewer options. However, it is in theory less likely to find as good of a solution as the unrestricted greedy approach. The method is explained using pseudocode in Listing 5.4.

```
restricted_greedy_insertions(solution, requests_to_insert):
  shuffle the requests in requests_to_insert

  for request in requests_to_insert:
    while request is not fully served:
      single_insertion(solution, request)
```

Listing 5.4: *When doing restricted greedy insertions, this procedure is done instead of the one in the function named insertions. Here, the order in which the requests are inserted is predetermined and random.*

5.5 Checking Feasibility

In the process of checking the feasibility of a route, every adjacent pair of visits in the route are iterated over. For each such pair, the following is done: First, it is checked that there exists an arc between the nodes associated with the visits. Then the elapsed time is updated by adding the travel time and service time at the second visit, and then it is checked if the time horizon has been exceeded. Next, the total cargo mass and total cargo volume is updated based on the mass change in the second visit, and it is checked whether the mass capacity or the volume capacity of the vehicle is exceeded. Furthermore, the energy balance of the vehicle is calculated by subtracting the energy consumption of traveling between the nodes of the currently considered visits, and it is checked if the energy balance goes below zero. Additionally, it is checked if the vehicle has arrived at a node where charging cannot be done with a SoC level below the value of a user-defined parameter named *min arrival soc*, which can for example be set to 20%. This last condition is not specified in the mathematical model, but helps avoiding situations where an electric vehicle gets stuck because it does not have enough energy to reach a nearby charging station when running the algorithm. Lastly, the charging done during the second visit is taken into account, and it is checked if the energy balance after charging exceeds the battery capacity of the vehicle. If any of the checks fails, the route is considered infeasible, but if it is infeasible due to constraints related to energy, there may be a possibility of repairing it.

5.6 Repairing Infeasible Routes by Adding Charging

When a route has been determined to be infeasible due to violating constraints related to energy, an algorithm is run that tries to repair the route. The algorithm initiates the energy balance to the battery capacity of the vehicle, assuming that the vehicle starts fully charged, and iterates over every pair of adjacent visits in the route. For each pair of visits, where the first visit is referred to as the current visit, and the second visit is referred to as the next visit, the following is done: First, charging at the current visit is added to the energy balance, and if the energy of the vehicle exceeds its battery capacity, the amount of energy charged is adjusted. Next, the energy consumption for traveling between the nodes is calculated and subtracted from the energy balance. If the energy consumption was negative, i.e. there was energy recuperation, the charging done before traveling to the next node is reduced if possible. Following this, it is checked if the next node was arrived at with a negative SoC, or if the SoC is below the value of the parameter *min arrival soc* and charging cannot be done at the next node. If this is the case, the energy balance is updated as if the current arc was never traveled, and then a charging path is searched for that starts at the current node and ends up at the next node. If such a path is found, the path is inserted into the current route, and the procedure continues. If at any point the next node could not be reached with an appropriate SoC, and no charging path could be found, the route could not be repaired and is deemed to be infeasible.

5.7 Finding a Charging Path

Before the ALNS procedure begins, charging paths are precomputed for each electric vehicle type. This is done by running an algorithm based on Dijkstra's algorithm once for each node in the graph where that node is the starting node. In this algorithm, the "shortest" paths from the given node to all other nodes in the graph are searched for, and these paths are referred to as charging paths as they go through nodes where the vehicle charges fully. The algorithm uses energy consumption as the "distance", and throughout the search, the battery energy of the vehicle upon arriving at a node is kept track off, making it possible to decide how much charging needs to be done to charge fully. During the search, only nodes where charging can be done are considered, and if a node cannot be reached with a non-negative SoC, that option is disregarded. Moreover, if charging is possible at the start node, the vehicle is charged there, which can reduce the number of visits in the charging path.

Once the search concludes, the shortest paths between the provided start node and the other nodes are extracted. More specifically, these are paths that go between the nodes such that the energy of the vehicle never becomes negative, charging to the max is done at every intermediate visit, and the energy consumption is lower than the other considered options. It should be noted that there may not exist a charging path between two nodes, for example if charging stations are sparse in the

problem instance.

After the charging paths have been precomputed, they are stored so that they can be used when trying to repair a route in the main algorithm. When such a situation arises, and a charging path between a node A and a node B is needed, the sequence of nodes from the precomputed charging path from A to B is utilized. As the vehicle may have a different SoC compared to the starting SoC that was used when precomputing the path, the charging amounts cannot be precomputed and are therefore calculated when the charging path is needed. This is done by iterating over the node sequence in the charging path, keeping track of the energy balance, and adding charging such that the vehicle charges fully at every stop. Once a proper path consisting of visits where charging amounts are specified has been obtained, it can be inserted into a route of an electric vehicle in the reparation process.

5.8 The ALNS Procedure

To be able to run ALNS, initial feasible routes are needed. These are obtained by starting with placeholder route for each vehicle that goes from the start home depot node to the end home depot node, and then using greedy insertion to insert all requests. ALNS starts with the solution that consists of these routes and is run for the selected number of iterations.

In each iteration, the accepted solution is started with and a removal method is chosen randomly using a roulette wheel approach and is then applied. Subsequently, an insertion method is randomly chosen and applied and the result is a new feasible solution. The objective value of this solution is compared to the objective value of the accepted solution, and if it is better it becomes the new accepted solution. When simulated annealing is used, a worse solution may also be accepted with a relatively low probability. If the current solution has lower objective value than the best known solution, the best solution is overwritten. After the insertions are done, it is checked if there are any infeasible routes, and if there are, those routes are repaired. This step exists because the following scenario can occur and needs to be dealt with for some routes in some iterations: at least one request is removed from a route and no request is inserted into that route, leaving it infeasible.

Before running ALNS, the number of iterations in a segment is defined by the user, and then each time that number of iterations have passed, the weights of the insertion heuristics and removal heuristics are updated. The removal and insertion heuristic weights are updated separately, using the same procedure. First, the success rate of each insertion or removal method is calculated by dividing the number of times the method led to a better solution in the last segment by the number of times it was used in the last segment. If the method was not used, its success rate will be set to 0. Then, the weight of each method, W_m, is updated in the following manner: $W_m = W_m \cdot (1 + \textit{success rate} \cdot \textit{max percentage increase})$, where *max percentage increase* is a user-defined value. Lastly, the weights of all methods are normalized, so that they sum to 1. Now, the heuristic methods that have contributed to finding better

solutions in past iterations have a higher probability of being selected with the roulette wheel approach in the coming iterations.

The algorithm continues destroying and repairing the accepted solution every iteration, and updating the weights upon the completion of each segment, and after the specified number of iterations have been completed, it terminates.

5.9 Applying Noise to Diversify the Search

Out of the four insertion methods used, only one of them is nondeterministic, while the remaining three are deterministic. Due to this, there is a risk that the same solutions may be found many times while running the algorithm, which makes the search less effective. In an attempt to counteract this, the search can be diversified by adding noise to the values that are compared when determining which insertion is best. In the algorithm developed for this thesis, noise is applied to the adjusted route cost deltas and regret values in the insertion procedure, if the parameter *initial max noise percentage* is set to a value greater than zero. The max noise percentage is set depending on the parameter value, and then during the algorithm run each route cost delta and regret value is multiplied by a noise factor, obtained by adding a random value in the range [*-max noise percentage, max noise percentage*] to 1. Additionally, a parameter named *max noise percentage increase* has been introduced. When this parameter is set to a positive value, the max noise percentage increases by the desired amount after each segment in which no better solution was found. Once a better solution is found, the max noise percentage is reset to its initial value. The idea is that the noise will help find more solutions, that may otherwise have been missed, and the noise increase mechanism is designed to diversify the search more and more in the situation that the algorithm gets stuck, possibly in a local optimum, and fails to find a better solution for many segments.

5.10 Simulated Annealing

Another strategy to diversify the search and avoid the situation where the algorithm gets stuck at a certain solution is to apply simulated annealing. When simulated annealing is used, there is a temperature T that affects the probability of a solution being accepted after it is obtained following the reinsertion of the requests removed in an iteration. The probability of a solution being accepted is $\exp\left(-\frac{f(s')-f(s)}{T}\right)$ where $f(s)$ is the objective value of the starting solution during the iteration, and $f(s')$ is the objective value of the new solution found in that iteration. The advantage of such a mechanism is that a worse solution may be accepted from which it is possible to reach a new best solution, whereas it may not have been possible to get to this new best solution by starting from the previous best solution.

Initially, a starting temperature is set, and then it gradually decreases with each iteration in accordance with the formula $T = T \cdot c$, where c is the cooling rate and is between 0 and 1. This means that worse solutions are more likely to be accepted

during the beginning of the search compared to the end. The initial temperature and the cooling rate are parameters that need to be set, but in this case this can be difficult due to the abstractness of the temperature value. Therefore, two different parameters are used in this thesis: *start temperature control parameter*, and *end temperature percentage*. The former parameter was introduced in [11] and works in the following manner: if its value is set to x, then the starting temperature will be set so that a solution that has a x% higher objective value compared to the initial solution will be accepted with 50% probability. The latter parameter is new in this thesis, and replaces the cooling rate parameter. If *end temperature percentage* is set to y, then the cooling rate will be set so that the temperature in the last iteration will be approximately y% of the initial temperature. With such parameters, adjusting the simulated annealing configuration becomes somewhat more intuitive.

6

Results

In this chapter, the results that were obtained when using the algorithm to solve different instances of the problem are presented.

6.1 Setup

This section contains information about problem instances, the fleet, and algorithm parameters that were used when running the algorithm. The algorithm has been implemented in Python, and the version 3.13.3 was used during the runs, with the default interpreter. It was run on a PC with a 12th Gen Intel® Core™ i5-1250P 1.70 GHz processor, and 16 GB RAM.

6.1.1 Problem Instances

The problem instances used have been created by utilizing data from a transportation company based in Europe that is a customer of Volvo Group. As the data spans many months, different instances were obtained by using data from different time periods.

An instance that is suitable for the algorithm has a home depot node, customer nodes that come in pickup and delivery pairs, and a set of standalone charging station nodes. The customer nodes where pickup is done support the loading types OBW and AST, and the ones where delivery is done only support the loading type OBW. Another thing to note is that each instance has a single home depot that has been duplicated to get a start depot node and an end depot node.

The home depot node and the customer nodes were obtained from the previously mentioned data, while the charging station nodes were obtained from a different source: a database containing existing charging stations and charging stations that are planned to be built in the future. Furthermore, the assumption that charging can be done at customer nodes has been made to facilitate the use of electric vehicles.

The instances referred to in this section are summarized in the Table 6.1.

Instance	No. Depot Nodes	No. Customer Nodes	No. Pure Charging Nodes	Total No. Nodes	Total Cargo Mass (kg)
01	2	30	10	42	378 685
02	2	18	62	82	219 474
03	2	25	66	93	320 652

Table 6.1: *Details regarding the three problem instances used in this master thesis project.*

6.1.2 Fleet Composition

The fleet consists of eight vehicles, of which the first seven are BEVs. Each BEV is one of two different models. Vehicles one through four are units of a model that has shorter range, but higher mass capacity. Vehicles five through seven are of a different model that has longer range, but lower mass capacity. The eight vehicle is a diesel vehicle that has the highest mass capacity in the fleet. The single diesel vehicle is present in the fleet to handle situations where it is not possible to serve certain customers with electric vehicles due to lacking charging infrastructure. To discourage the use of needlessly many vehicles, every vehicle in the fleet has an associated cost, referred to as deployment cost, that is added to the objective value when the vehicle is used to serve at least one customer. This cost is twice as high for the diesel vehicle, so that it is only used when necessary. The motivation for this is that a customer may want reduce their carbon footprint and transition to using only electric vehicles, but is ready to use a diesel vehicle if their operations cannot be done without it due to a lack of charging stations.

More detailed information about the vehicles in the fleet is presented in Table 6.2.

ID	Type	Battery Capacity (kWh)	Charging Power (kW)	Mass Capacity (tons)	Volume Capacity (m^3)	Loading Types	Deployment Cost (€)
1	Electric 1	~400	250	~27	~28	OBW, AST	~160
2	Electric 1	~400	250	~27	~28	OBW, AST	~160
3	Electric 1	~400	250	~27	~28	OBW, AST	~160
4	Electric 1	~400	250	~27	~28	OBW, AST	~160
5	Electric 2	~590	350	~25	~28	OBW, AST	~160
6	Electric 2	~590	350	~25	~28	OBW, AST	~160
7	Electric 2	~590	350	~25	~28	OBW, AST	~160
8	Diesel	N/A	N/A	~29	~28	OBW, AST	~320

Table 6.2: *Details regarding the eight vehicles in the fleet used in the algorithm runs. The ~ sign indicates that an approximate value has been used instead of an exact value. The exact values are trade secrets and cannot be shared in this thesis.*

6.1.3 Base Algorithm Configuration

While some parameters have been changed between runs in this section, other parameter values remain the same throughout. The density of the products transported has been set to the equal to the density of concrete. The segment size has been set to 25, meaning that the weights of the heuristic are updated every 25 iterations, as well as the max noise value if noise is used. The parameter *max weight percentage* has been set to 1, meaning that weight of an insertion or removal method can at most increase by 100% before normalization, after a segment is completed. The removal percentage range has been set from 0.1 to 0.5, meaning that for removal methods, excluding single route removal, a random value between 10% and 50% is selected and approximately this proportion of the requests is removed. The parameter *worst removal determinism rate* has been set to 0.66, to introduce some stochasticity to the worst removal method in attempt to reduce the risk of finding the same solutions multiple times. The minimum arrival SoC has been set to 0.2, meaning that the sequence of nodes in the charging paths between nodes is precalculated with the assumption that the vehicle begins with a SoC of 20%. Finally, an additional charging rule has been applied so that charging is always done when loading or unloading if possible, because doing these actions in parallel is cost

effective.

6.1.4 Algorithm Variants

In the first part of the results section, three different algorithm variants are used during the runs, with the goal of being able to compare the effect of adding noise, and simulated annealing. In the second part of the results section, a fourth variant with both noise and simulated annealing was used in longer runs as it is more important to diversify the search when the number of iterations is greater.

Basic

In this variant, no noise is added, and simulated annealing is not used. As such, a new solution is only accepted if it is the best one found so far.

With Noise

In this variant, noise is added. The parameter *initial max noise percentage* is set to 10%, and the parameter *max noise percentage increase* is set to 10%.

With Simulated Annealing

In this variant, simulated annealing is used. The temperature control parameter is set to 2.5%, and *end temperature percentage* is set 1%.

With Noise and Simulated Annealing

In this variant, noise is added and simulated annealing is used. The parameter *initial max noise percentage* is set to 10%, *max noise percentage increase* is set to 10%, the temperature control parameter is set to 2.5%, and *end temperature percentage* is set to 1%.

6.2 Initial Solutions

To get initial solutions, necessary to start running the algorithm, greedy insertions were done into routes that only contained the start depot node, and end depot node. As no noise was added, and the greedy insertion method is deterministic, every single run for a specific instance started with the same initial solution.

Information about the initial solutions obtained for the three instances is provided in Table 6.3.

Instance	Time Horizon (h)	Objective Value (€)	Vehicles Used
01	36	3882.19	3, 4, 8
02	72	7302.66	6, 7, 8
03	48	7375.40	3, 4, 7, 8

Table 6.3: *Details regarding the initial solution to each instance, obtained using a greedy approach.*

6.3 Shorter Algorithm Runs: Comparing Variants

Each problem instance was solved using different variations of the algorithm. The number of iterations, and the random seed were changed to be able to see how these parameters affect the objective value of the solution that the algorithm finds.

First, 27 algorithm runs were conducted on instance 01, where the basic variant, the variant with noise, and the variant with simulated annealing were used. The time horizon was set to 36 hours, the number of iterations was set to 250, 500, or 1000, and three different random seeds were used for each distinct number of iterations for each of the three algorithm variations. The results obtained in these runs are shown in Table 6.4.

Algorithm Variant	Random Seed	No. Iterations	Time Horizon (h)	ALNS Runtime (h:mm:ss)	Best Objective Value (€)	Vehicles Used
Basic	1	250	36	0:03:00	3568.58	3, 4, 8
Basic	1	500	36	0:06:03	3568.58	3, 4, 8
Basic	1	1000	36	0:16:40	3568.58	3, 4, 8
Basic	2	250	36	0:02:28	3760.00	3, 4, 8
Basic	2	500	36	0:04:57	3717.60	3, 4, 8
Basic	2	1000	36	0:10:25	3717.60	3, 4, 8
Basic	3	250	36	0:03:46	3553.82	3, 4, 8
Basic	3	500	36	0:10:01	3537.48	3, 4, 8
Basic	3	1000	36	0:09:43	3537.48	3, 4, 8
With Noise	1	250	36	0:03:04	3575.46	3, 4, 8
With Noise	1	500	36	0:08:06	3570.41	3, 4, 8
With Noise	1	1000	36	0:13:40	3570.41	3, 4, 8
With Noise	2	250	36	0:02:49	3809.54	3, 4, 8
With Noise	2	500	36	0:06:07	3681.08	3, 4, 8
With Noise	2	1000	36	0:13:07	3558.53	3, 4, 8
With Noise	3	250	36	0:03:32	3753.64	3, 4, 8
With Noise	3	500	36	0:06:49	3615.84	3, 4, 8
With Noise	3	1000	36	0:14:06	3611.75	3, 4, 8
With SA	1	250	36	0:03:05	3694.54	3, 4, 8
With SA	1	500	36	0:04:37	3615.85	3, 4, 8
With SA	1	1000	36	0:08:41	3729.69	3, 4, 8
With SA	2	250	36	0:02:42	3555.11	3, 4, 8
With SA	2	500	36	0:04:28	3556.88	3, 4, 8
With SA	2	1000	36	0:10:26	3580.99	4, 7, 8
With SA	3	250	36	0:01:58	3785.33	2, 3, 4, 8
With SA	3	500	36	0:04:37	3633.22	3, 4, 8
With SA	3	1000	36	0:08:33	3571.99	3, 4, 8

Table 6.4: *Summary of the shorter algorithm runs done on instance 01.*

In these runs, the solution with the lowest objective value was found when the basic algorithm variant was used together with the random seed 3, and when the number of iterations was set to 500 and 1000. A figure illustrating how the search progressed in the aforementioned run with 500 iterations (Figure 6.1) is shown below, followed by a visualization of the solution that presents the order in which each vehicle traveled, loaded and unloaded cargo, and in the case of the electric vehicles, charged.

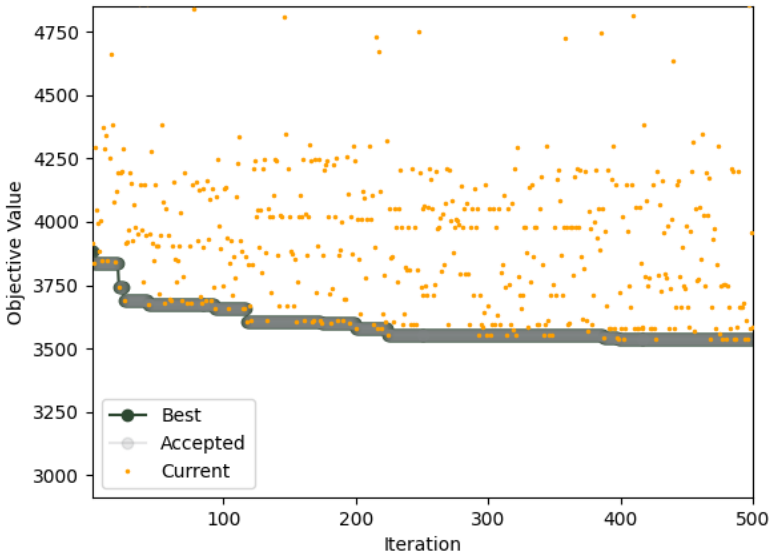

Figure 6.1: *The progression of the search in the run where the basic variant variant was used, the random seed was set to 3, and the number of iterations was set to 500. This run was one of two runs that yielded the lowest objective value for instance 01.*

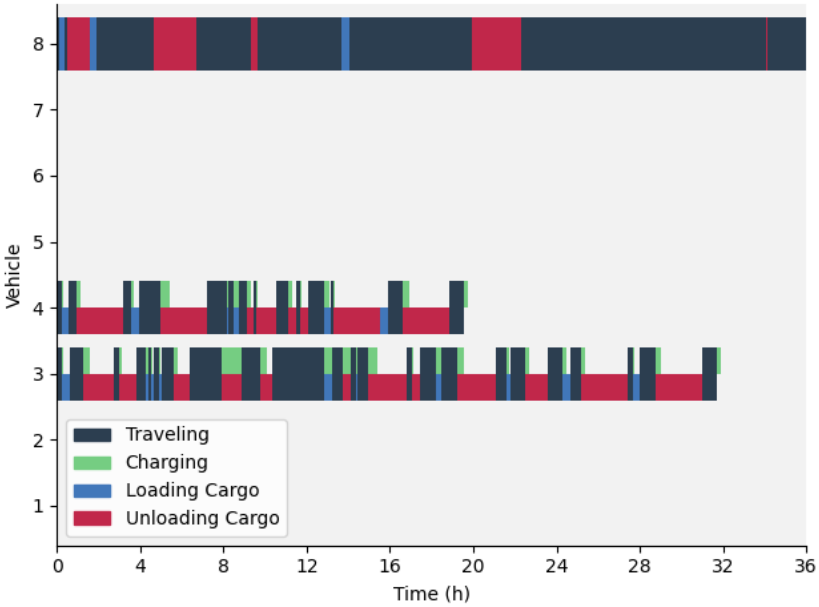

Figure 6.2: *A visualization of the best solution found in the shorter runs on instance 01. It was obtained in the run where the basic variant variant was used, the random seed was set to 3, and the number of iterations was set to 500.*

After these runs, similar runs were done on instance 02, with the difference being that the time horizon was set to 72 hours. In Table 6.5, the result, including the

runtime of ALNS, the objective value of the best solution, and the vehicles used can be found for each of the 27 runs.

Algorithm Variant	**Random Seed**	**No. Iterations**	**Time Horizon (h)**	**ALNS Runtime (h:mm:ss)**	**Best Objective Value (€)**	**Vehicles Used**
Basic	1	250	72	0:00:44	6707.53	6, 7, 8
Basic	1	500	72	0:01:38	6413.94	6, 8
Basic	1	1000	72	0:04:12	6413.94	6, 8
Basic	2	250	72	0:00:45	6690.81	6, 7, 8
Basic	2	500	72	0:02:02	6690.81	6, 7, 8
Basic	2	1000	72	0:03:18	6266.59	7, 8
Basic	3	250	72	0:00:47	6418.28	7, 8
Basic	3	500	72	0:02:24	6418.28	7, 8
Basic	3	1000	72	0:04:48	6418.28	7, 8
With Noise	1	250	72	0:00:45	6707.53	6, 7, 8
With Noise	1	500	72	0:01:33	6266.59	6, 8
With Noise	1	1000	72	0:03:03	6266.59	6, 8
With Noise	2	250	72	0:00:53	6414.64	7, 8
With Noise	2	500	72	0:01:39	6414.64	7, 8
With Noise	2	1000	72	0:03:43	6400.79	7, 8
With Noise	3	250	72	0:00:44	6418.28	7, 8
With Noise	3	500	72	0:01:34	6418.28	7, 8
With Noise	3	1000	72	0:04:05	6418.28	7, 8
With SA	1	250	72	0:00:52	6706.84	6, 7, 8
With SA	1	500	72	0:01:51	6266.59	6, 8
With SA	1	1000	72	0.03:42	6266.59	6, 8
With SA	2	250	72	0:00:44	6266.59	7, 8
With SA	2	500	72	0:01:36	6791.57	3, 4, 8
With SA	2	1000	72	0:03:16	6690.44	6, 7, 8
With SA	3	250	72	0:00:46	6266.59	7, 8
With SA	3	500	72	0:01:30	5994.64	4, 8
With SA	3	1000	72	0:03:35	6734.51	4, 7, 8

Table 6.5: *Summary of the shorter algorithm runs done on instance 02.*

Out of the runs on instance 02, the run where the variant with simulated annealing was used, with the random seed set to 3, and the number of iterations set to 500, resulted in the best solution, being the only one with an objective value under 6000

€. The progression of the search for this run, and a visualization of the solution are shown in Figure 6.3 and Figure 6.4 respectively.

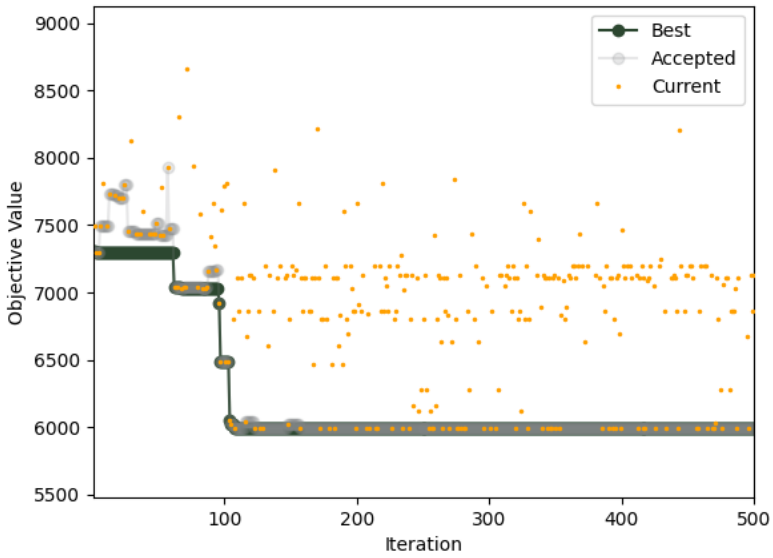

Figure 6.3: *The progression of the search in the run where the simulated annealing variant variant was used, the random seed was set to 3, and the number of iterations was set to 500. This run yielded the lowest objective value for instance 02.*

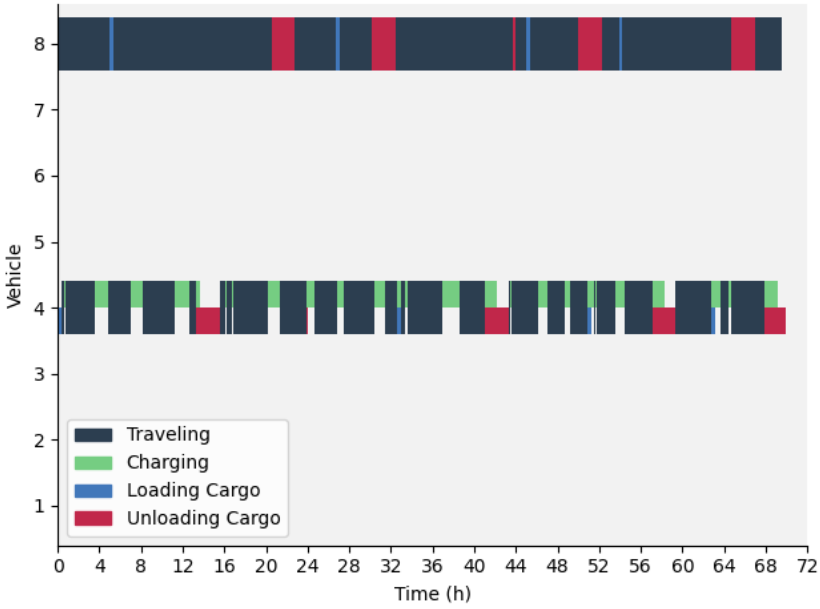

Figure 6.4: *A visualization of the best solution found in the shorter runs on instance 02. It was obtained in the run where the simulated annealing variant variant was used, the random seed was set to 3, and the number of iterations was set to 500.*

Next, 27 runs were performed on instance 03 in a similar manner as was done for the preceding two instances. In this case, the time horizon was set to 48 hours. The complete results for these runs are presented in Table 6.6.

Algorithm Variant	**Random Seed**	**No. Iterations**	**Time Horizon (h)**	**ALNS Runtime (h:mm:ss)**	**Best Objective Value (€)**	**Vehicles Used**
Basic	1	250	48	0:01:48	7184.07	3, 4, 7, 8
Basic	1	500	48	0:03:54	7184.07	3, 4, 7, 8
Basic	1	1000	48	0:07:25	7184.07	3, 4, 7, 8
Basic	2	250	48	0:01:52	7189.18	3, 4, 7, 8
Basic	2	500	48	0:03:45	7189.18	3, 4, 7, 8
Basic	2	1000	48	0:07:49	7189.18	3, 4, 7, 8
Basic	3	250	48	0:01:48	7120.51	3, 4, 7, 8
Basic	3	500	48	0:03:39	7120.51	3, 4, 7, 8
Basic	3	1000	48	0:07:17	7120.51	3, 4, 7, 8
With Noise	1	250	48	0:01:57	7184.07	3, 4, 7, 8
With Noise	1	500	48	0:03:49	7184.07	3, 4, 7, 8
With Noise	1	1000	48	0:08:14	7184.07	3, 4, 7, 8
With Noise	2	250	48	0:01:40	7189.18	3, 4, 7, 8
With Noise	2	500	48	0:03:22	7189.18	3, 4, 7, 8
With Noise	2	1000	48	0:06:35	7189.18	3, 4, 7, 8
With Noise	3	250	48	0:02:10	7184.07	3, 4, 7, 8
With Noise	3	500	48	0:04:59	7184.07	3, 4, 7, 8
With Noise	3	1000	48	0:09:16	7184.07	3, 4, 7, 8
With SA	1	250	48	0:02:06	7184.07	3, 4, 7, 8
With SA	1	500	48	0:04:48	7200.62	3, 4, 7, 8
With SA	1	1000	48	0:07:28	6944.64	3, 6, 7, 8
With SA	2	250	48	0:01:38	7082.68	2, 4, 7, 8
With SA	2	500	48	0:05:38	6975.93	3, 4, 7, 8
With SA	2	1000	48	0:09:49	6953.60	3, 4, 7, 8
With SA	3	250	48	0:01:35	7184.07	3, 4, 7, 8
With SA	3	500	48	0:03:54	7200.62	3, 4, 7, 8
With SA	3	1000	48	0:08:40	6944.64	3, 6, 7, 8

Table 6.6: *Summary of the shorter algorithm runs done on instance 03.*

Looking at the table, it can be observed that two runs yielded the best solution: the runs where simulated annealing was used and the number of iterations was set to

1000, and the random seed was set to 1 and 3. The progression of the search in the previously mentioned instance with the random seed set to 3 is shown in Figure 6.5, and following this, a visualization of the best solution found in this run is shown in Figure 6.6.

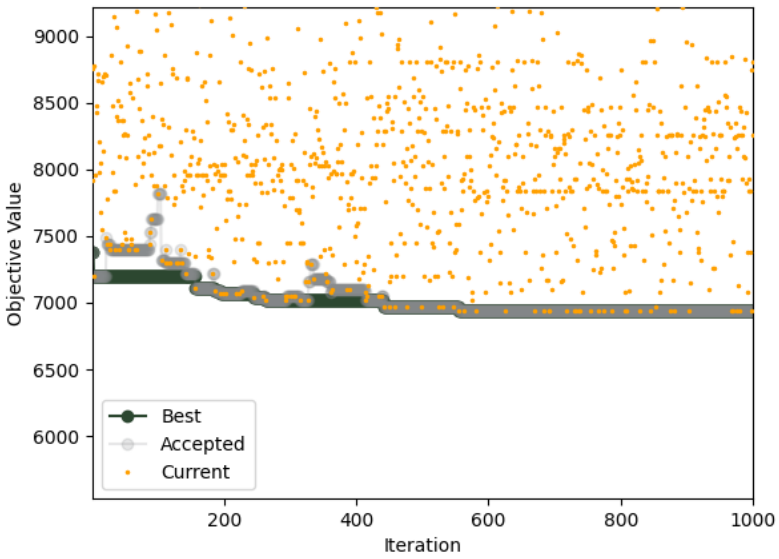

Figure 6.5: *The progression of the search in the best shorter run on instance 03. In this run, the simulated annealing variant variant was used, the random seed was set to 3, and the number of iterations was set to 1000.*

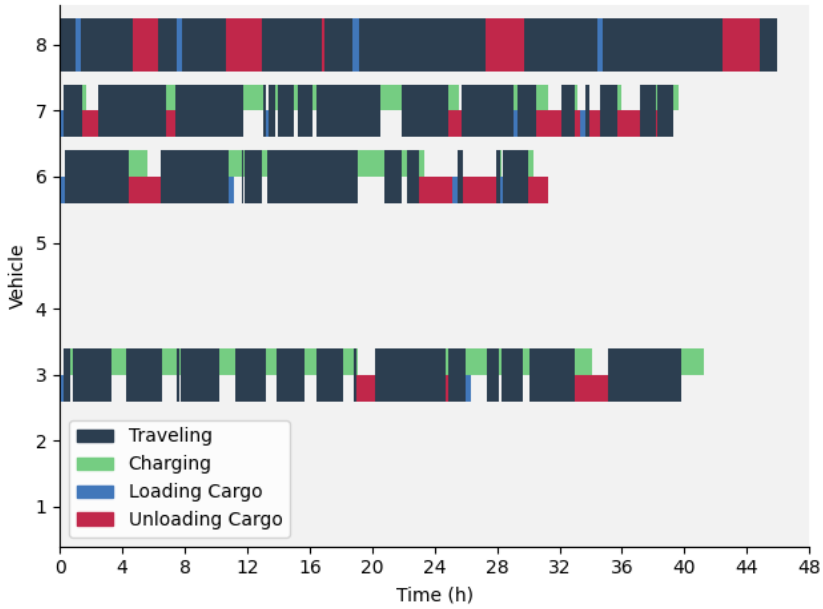

Figure 6.6: *A visualization of the solution found in one of the most successful shorter runs on instance 03. In this run, the simulated annealing variant variant was used, the random seed was set to 3, and the number of iterations was set to 1000.*

6.4 Longer Algorithm Runs: Searching for Better Solutions

Subsequently to the shorter runs, the algorithm was run for a considerably greater number of iterations to solve each instance and to see what solutions the algorithm is capable of reaching. In these runs, the variation with noise and simulated annealing was used. This variation was chosen because it is more important to diversify the search in such longer runs to avoid wasting hundreds of iterations by finding the same solutions over and over, and adding noise and simulated annealing should in theory help avoid this problem. The results obtained after performing a run on each instance with 10 000 iterations is presented in Table 6.7.

Instance	Random Seed	No. Iterations	Time Horizon (h)	ALNS Runtime (h:mm:ss)	Best Objective Value (€)	Vehicles Used
01	1	10 000	36	1:45:42	3517.63	3, 4, 8
02	1	10 000	72	0:33:23	5950.73	7, 8
03	1	10 000	48	0:59:27	6783.75	3, 4, 7, 8

Table 6.7: *Summary of the three longer runs that were done on the instances. For each instance, the algorithm was run once for 10 000 iterations using the variant with noise and simulated annealing.*

The run on instance 01 took approximately 1 hour and 46 minutes, and progressed as shown in Figure 6.7.

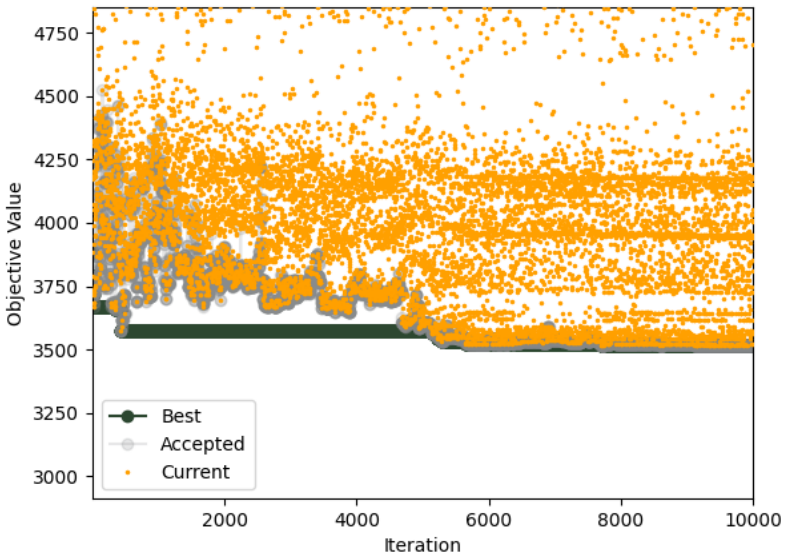

Figure 6.7: *A plot showing the objective value obtained in each iteration, and how the objective value of the accepted and best solutions changed over time in the longer run on instance 01.*

The best solution found for instance 01 in this run is better than all solutions found for this instance in the previous shorter runs. It is visualized below, in Figure 6.8.

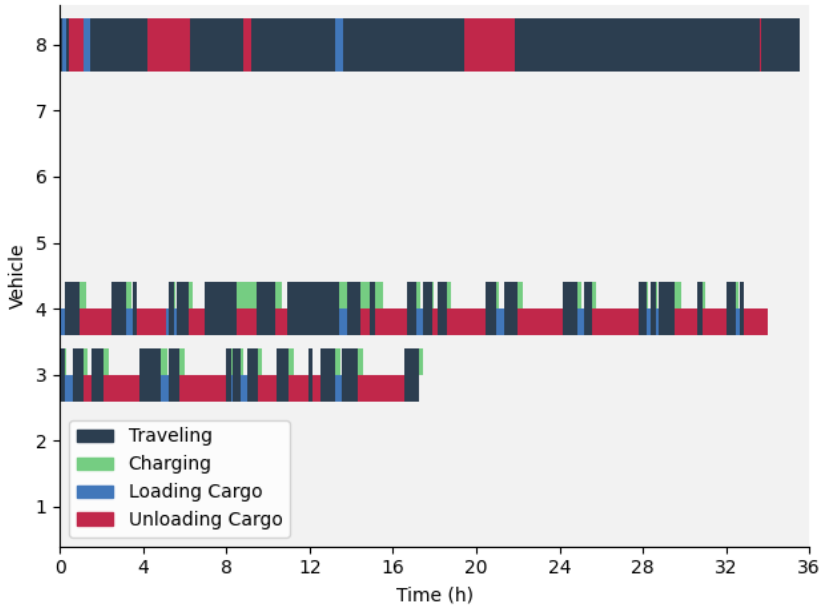

Figure 6.8: *A visualization of the best solution found for instance 01 after performing the longer run.*

The weights for the insertion methods and removal methods, which indicate how successful the methods were, were distributed as shown in Figure 6.9 after 10 000 iterations in the run on instance 01.

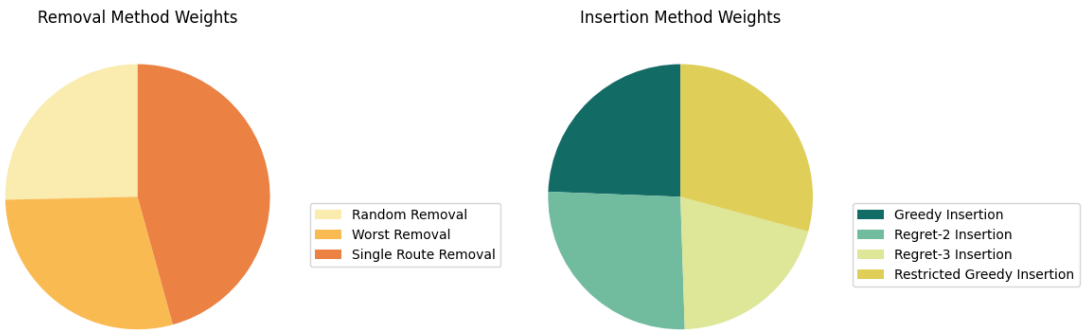

Figure 6.9: *The distribution of the weights of the removal methods, and the weights of the insertion methods after the longer run on instance 01.*

It the figure above it can be observed that the single route removal method has been rewarded much more than the other removal methods in the run on instance

01. Additionally, it can be observed that the different insertion methods have similar weights meaning that no one insertion method dominated.

In the run on instance 02, a new best solution was found. The progression of the search in this run and a visualization of the best solution found, respectively, are presented in Figure 6.10 and Figure 6.11 respectively.

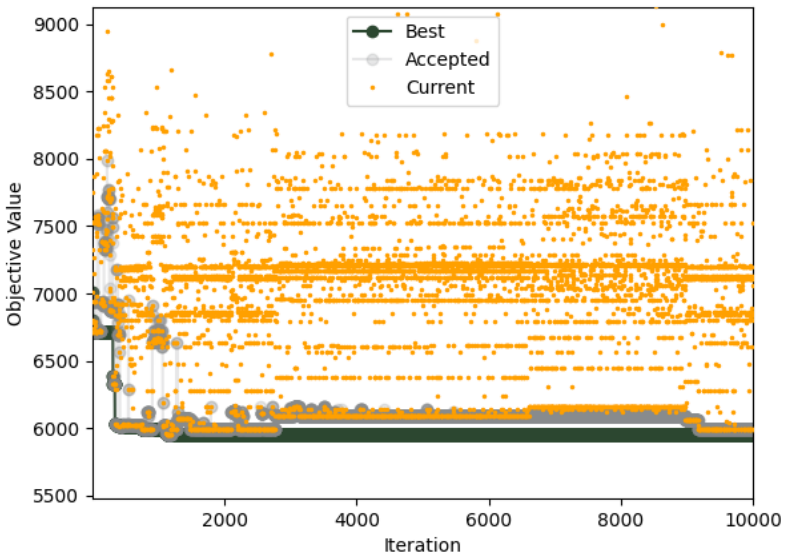

Figure 6.10: *A plot showing the objective value obtained in each iteration, and how the objective value of the accepted and best solutions changed over time in the longer run on instance 02.*

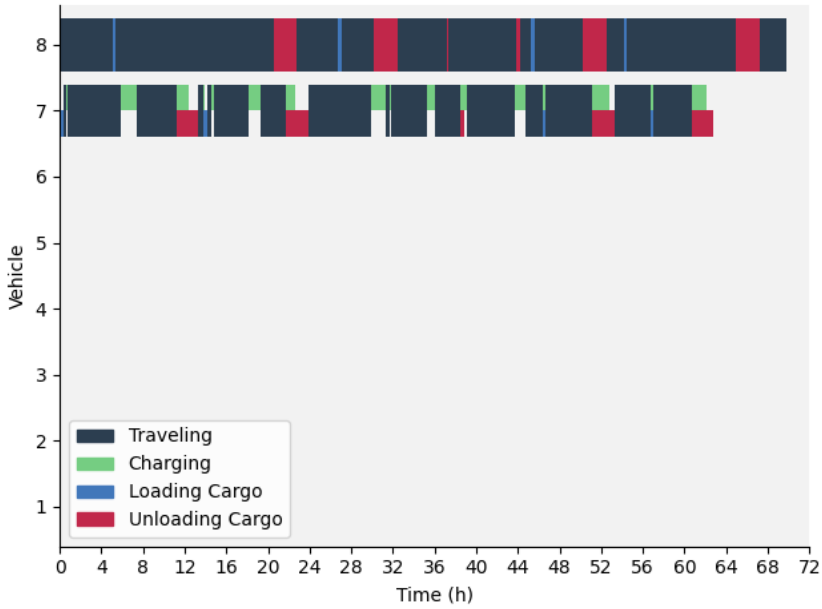

Figure 6.11: *A visualization of the best solution found for instance 02 after performing the longer run.*

After the longer run on instance 02, the heuristic weights were distributed in the manner shown in Figure 6.12.

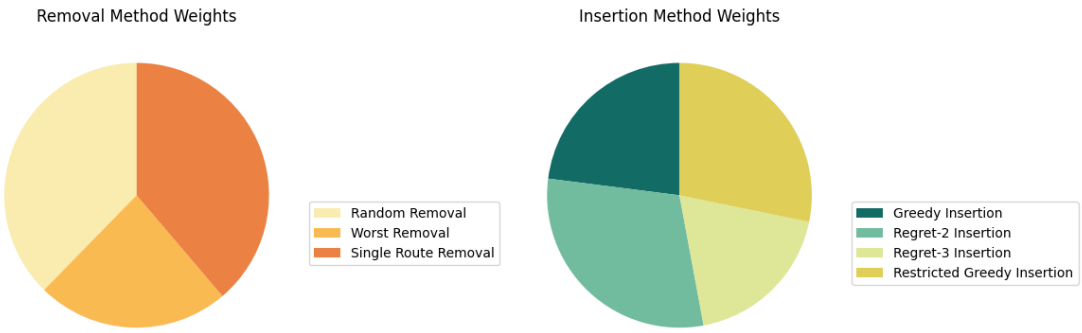

Figure 6.12: *The distribution of the weights of the removal methods, and the weights of the insertion methods after the longer run on instance 02.*

In the figure above, it can be seen that the worst removal method seemed to be less effective in the run on instance 02. The weights of the insertion methods are, as

seen before, distributed more equally, but it is noticeable that the regret-3 insertion method was rewarded less in this case.

The run on instance 03 was also successful as a new best solution for the instance was found in this case as well. A figure showing how the search progressed (Figure 6.13), and a visualization of the solution (Figure 6.14) are shown below.

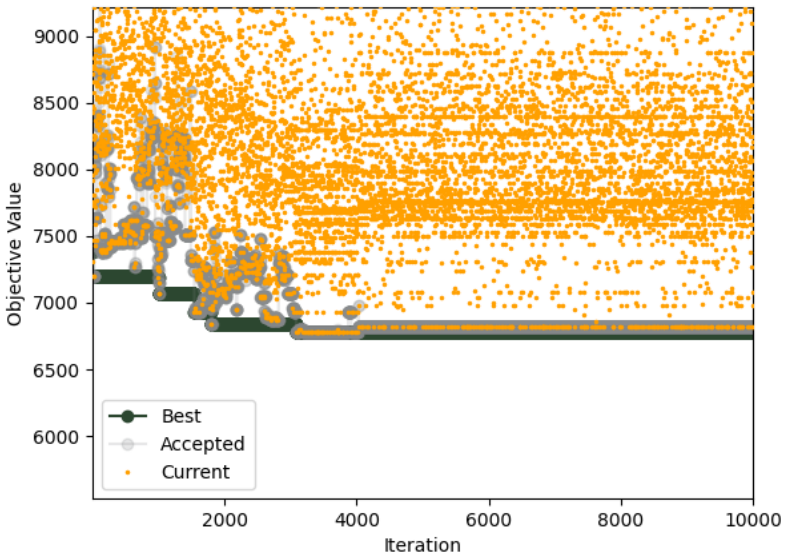

Figure 6.13: *A plot showing the objective value obtained in each iteration, and how the objective value of the accepted and best solutions changed over time in the longer run on instance 03.*

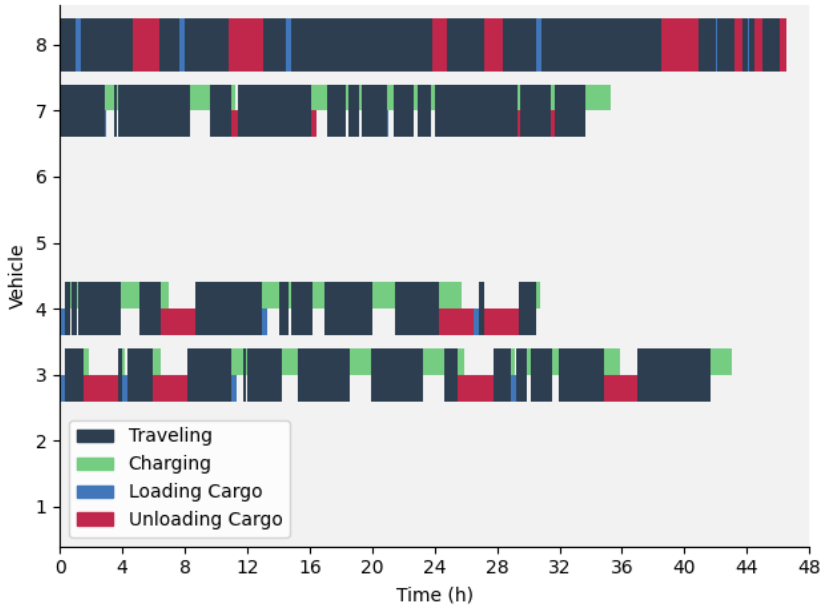

Figure 6.14: *A visualization of the best solution found for instance 03 after performing the longer run.*

The weights of the removal methods and insertion methods after the longer run on instance 03 are visualized in Figure 6.15.

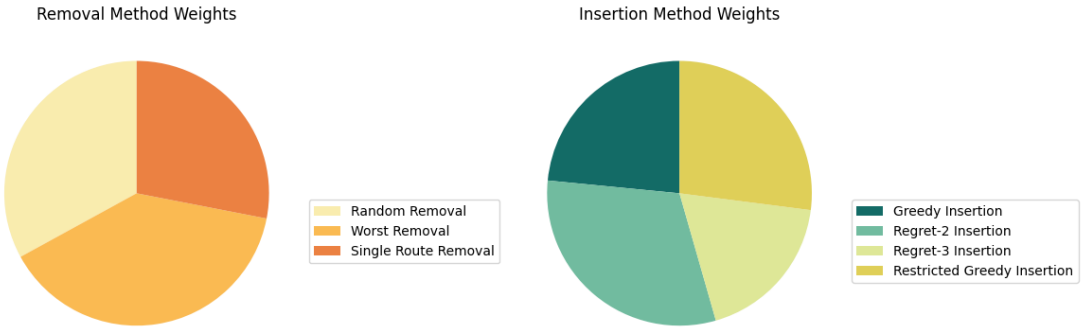

Figure 6.15: *The distribution of the weights of the removal methods, and the weights of the insertion methods after the longer run on instance 03.*

In the figure above, it can be seen that the worst removal method was rewarded the most in this run and seems to have performed the best, while the insertion method weights were roughly equal, with the weight of the regret-2 insertion method being

somewhat greater and the weight of the regret-3 insertion method being somewhat smaller.

7

Discussion

In this chapter different aspects related to the results are discussed, and the limitations of the algorithm as well as ideas regarding future work are presented.

7.1 The Effect of Noise and Simulated Annealing

In the first part of the results, three different variants were used in the algorithm runs to investigate the effect of applying noise and using simulated annealing.

When looking at the variant with noise, and its runs on instance 01, solutions with better objective values were only found in two out of the nine runs compared to the basic variant. For the runs on instance 02, better solutions were found in four out of the nine runs compared to the basic variant, and identical solutions were found in the other runs, excluding one run where a higher objective value was ultimately found. For the runs on instance 3, the same solutions that were found when no noise was used were found in six cases, and in the remaining three cases worse solutions were found.

Overall the effect of applying noise was inconsistent. It seems that the aforementioned variant helps diversify the search, but the solutions that would otherwise not have been found are in many cases not better than the solutions found when no noise is added. Thus, it appears that solely adding noise does not consistently lead to finding better solutions, at least for the three available instances.

When looking at the simulated annealing variant, better solutions were found in three out of nine runs on instance 01 compared to the basic variant, the ones where the random seed 2 was used. For the runs on instance 02, the variant with simulated annealing resulted in better solutions in six out of nine runs compared to the basic variant. For instance 03, a similar effect of applying simulated annealing can be observed. In this case, the results were better to the solutions found by the basic variant in five out of nine runs, and equal in one run.

The effect of simulated annealing was beneficial in most runs on instances 02 and 03, and in some runs on instance 01. Moreover, out of the runs in the first part of the

results, the variant with simulated annealing yielded the best solution to instance 02, and instance 03. Due to this, it seems worthwhile to use simulated annealing even though it does not guarantee a better solution in every run compared to the basic variant.

Some other things to note following these results is that all three variants were essentially equally fast, which was expected. Furthermore, it is clear that the choice of random seed can have an effect on the result when algorithm runs with up to 1000 iterations are performed.

7.2 Gauging the Quality of the Solutions

In this thesis, the goal of being able to solve realistic large-scale instances based on data has been fulfilled. The ALNS framework turned out to be a good fit for solving the problem at hand and solutions can be found within minutes with the developed algorithm. In the longer runs, the algorithm found better solutions compared to the shorter runs, which is a testament that the search progresses effectively.

The solutions that the algorithm outputs look reasonable, but it is difficult to objectively say how good the solutions are as there exist no reference solutions for these instances and this problem formulation, and the problem is too complex to be solved using MILP within a reasonable amount of time.

What can be done, however, is to compare the best solutions output by the algorithm with the initial solutions that were created using a greedy approach. Compared to the initial solution for instance 01, the best solution found in the shorter runs had an objective value that is approximately 9.7% lower, and the solution found in the longer run had an objective value that is approximately 10.4% lower. For instance 02, the best solution in the shorter runs yielded a 21.8% lower objective value, and the solution found in the longer run yielded a 22.7% lower objective value in comparison with the initial solution. For instance 03, the best solution found in the shorter runs had a 6.3% lower objective value, and the solution found in the longer run had a 8.7% lower objective value, compared to the initial solution. While these results are satisfactory, it is unclear how close the best objective value found for each instance is to the global optimum.

7.3 Analyzing the Vehicles Used

For the different instances, different types of vehicles are used in the best solutions. In the solution for instance 01, two electric vehicles are used of the type with lower battery capacity, and higher mass capacity. In contrast, the electric vehicle model with higher battery capacity and lower mass capacity is preferred in the best solution for instance 02. In the best solution for instance 03, both types of electric vehicles are used. This indicates that both electric vehicle models have their merits and which one is better to use depends on the demands of the customers, and how the

customer nodes are laid out geographically.

Furthermore, it should be pointed out that every solution presented in the results section used vehicle nr. 8, i.e. the single diesel vehicle in the fleet. Since the deployment cost of the diesel vehicle was higher, this suggests that while BEVs are viable, it is necessary to use a diesel vehicle to serve some customers in all three problem instances. The reason for this must be that the charging infrastructure is lacking, or more specifically that there exist regions where there are no charging stations but are needed. As such, the conclusion is that more charging stations need to be built, and that they need to be placed strategically so that electric vehicles can be used for transportation without impediments in the future.

7.4 Limitations of the Algorithm

While the algorithm has given adequate results, it has some limitations that are worth mentioning.

First, it seems unlikely that the algorithm will find the globally optimal solution, partly due to the fact that charging is handled with a simplistic rule based approach where charging stops are inserted when the next node cannot be reached and charging is done to the max each time it is performed. This charging strategy is not ideal, but on the other hand charging needs to be added very often when it is incorporated into ALNS, and it is not viable to select an intricate but slow method as it would become a bottleneck that slows down the algorithm. Moreover, the split pickups and deliveries are handled by taking as much cargo as possible each time. This seems logical, but it is unclear if this approach enables the algorithm to reach the global optimum.

Second, when running the algorithm, many parameters need to be set and this can be a difficult task. For example, the removal percentage range, parameters related to simulated annealing and noise, the number of iterations, the minimum arrival SoC, and the deployment costs for the vehicles need to be set. The deployment cost can be particularly hard to decide on as setting it too low leads to the usage of too many vehicles, while setting it to high essentially prevents the algorithm from engaging more vehicles unless no feasible insertions exist for the vehicles that are already in use.

Third, the algorithm needs a feasible solution to start. While such a solution can often be obtained by using any one of the insertion methods to insert all of the requests in an instance, there may be cases where it is not possible to get an initial solution that is feasible using this approach, and then the algorithm cannot be run.

7.5 Potential for Future Work

The algorithm developed in this thesis has many parameters, and to achieve the best results they may have to be adjusted further. For example, for the simulated annealing variant, the temperature control parameter was set to 2.5%, and the end temperature percentage to 1%, and they were set in this manner because it seemed reasonable. However, there is no guarantee that these are the best values for these parameters, and this is just a single example. To set these parameters in a better way, a technique similar to grid search in machine learning could be employed in the future.

Another thing to think about is how the algorithm should be used to get the best possible solution. The results showed that running the algorithm for a greater number of iterations leads to better solution being found. At the same time, it is clear that the value of the random seed has an impact on the solution. Therefore, one way of using the algorithm would be to do a long run with simulated annealing, but an alternative could be to do multiple shorter runs with differing random seeds, maybe even in parallel. The latter approach would be a different way of handling the problem of the algorithm getting stuck at a solution that may be locally optimal. To increase the probability of obtaining heterogenous solutions in the parallel runs, different initial solutions could even be used by using the nondeterministic restricted greedy insertions method.

Moreover, the process of adding charging during the ALNS procedure could potentially be improved further, for example by avoiding unnecessarily short charging visits, not forcing a vehicle to charge fully during every charging stop, or by considering aspects other than energy consumption when determining the charging paths between nodes.

Finally, while the problem definition in this thesis is relatively complicated, there is still potential for further extensions. For example, time windows and waiting times at charging stations are some aspects that have been explored in other studies, but were not considered in this thesis.

8

Conclusion

The objective of the thesis was to develop a method for solving large-scale instances of the presented variation of FSMVRP in a reasonable amount of time, and with the developed metaheuristic algorithm, this objective has been fulfilled. The ALNS framework turned out to be a good choice for handling the problem, and the most challenging aspects, split pickups and deliveries, and the charging of electric vehicles could be dealt with. More specifically, a flexible approach where the vehicles pick up and deliver as much cargo as possible was proposed, and the insertion methods were adjusted to support this way of solving the problem. As for charging, an approach using so called precomputed charging paths was introduced, which is practical, and beneficial from the perspective of computational efficiency. Utilizing these ideas, together with the concept of ALNS, it was possible to implement algorithm that managed to find satisfactory solutions to three realistic problem instances. When running the algorithm to solve the instances, variants with noise and simulated annealing were used in addition to a basic variation. While the effect of adding noise turned out to be mediocre, using simulated annealing showed greater potential. Going forward, further experimentation would be required to determine how the parameters should be set to achieve the best results, and the algorithm would have to be run on more instances to find the definitive configuration.

Bibliography

[1] Council of the European Union, *Council signs off on stricter co2 emission standards for heavy-duty vehicles*, Accessed: 2024-11-29, 2024. [Online]. Available: https://www.consilium.europa.eu/en/press/press-releases/2024/05/13/heavy-duty-vehicles-council-signs-off-on-stricter-co2-emission-standards/.

[2] G. Laporte and Y. Nobert, "Exact algorithms for the vehicle routing problem," in *Surveys in Combinatorial Optimization*, ser. North-Holland Mathematics Studies, vol. 132, North-Holland, 1987, pp. 147–184. DOI: https://doi.org/10.1016/S0304-0208(08)73235-3.

[3] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[4] S. Elatar, K. Abouelmehdi, and M. E. Riffi, "The vehicle routing problem in the last decade: Variants, taxonomy and metaheuristics," *Procedia Computer Science*, vol. 220, pp. 398–404, 2023, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2023.03.051.

[5] B. Golden, A. Assad, L. Levy, and F. Gheysens, "The fleet size and mix vehicle routing problem," *Computers & Operations Research*, vol. 11, no. 1, pp. 49–66, 1984, ISSN: 0305-0548. DOI: https://doi.org/10.1016/0305-0548(84)90007-8.

[6] O. H. Palmqvist, "Solving extended Electric Fleet Size and Mix VRPs using column generation: A computational approach to the EFSMVRP with multi-trips and split pick-ups and deliveries," Volvo Group Trucks Technology, Dept. of Mathematical Sciences, Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, May 2024.

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

[8] C. Archetti and M. Speranza, "A survey on matheuristics for routing problems," *EURO Journal on Computational Optimization*, vol. 2, no. 4, pp. 223–246, 2014, ISSN: 2192-4406. DOI: https://doi.org/10.1007/s13675-014-0030-7.

[9] J. Kallestad, R. Hasibi, A. Hemmati, and K. Sörensen, "A general deep reinforcement learning hyperheuristic framework for solving combinatorial opti-

mization problems," *European Journal of Operational Research*, vol. 309, no. 1, pp. 446–468, 2023, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2023.01.017.

[10] The Association of European Operational Research Societies, *VeRoLog Solver Challenge – Verolog*, Accessed: 2024-11-29. [Online]. Available: https://www.euro-online.org/websites/verolog/verolog-solver-challenge/.

[11] S. Røpke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation Science*, vol. 40, no. 4, pp. 455–472, 2006, Accessed: 2025-04-28. DOI: 10.1287/trsc.1050.0135. [Online]. Available: https://doi.org/10.1287/trsc.1050.0135.

[12] G. Clarke and J. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, no. 4, pp. 568–581, 1964.

[13] B. Golden and A. Assad, "Heuristics for the vehicle routing problem," *Vehicle routing: methods and studies*, pp. 33–56, 1985.

[14] C. Prins, "A simple and effective evolutionary algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 31, no. 12, pp. 1985–2002, 2004.

[15] F. Glover, "Tabu search: A tutorial," *Interfaces*, vol. 20, no. 4, pp. 74–94, 1990.

[16] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[17] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[18] G. Hiermann, J. Puchinger, S. Ropke, and R. F. Hartl, "The electric fleet size and mix vehicle routing problem with time windows and recharging stations," *European Journal of Operational Research*, vol. 252, no. 3, pp. 995–1018, 2016. DOI: 10.1016/j.ejor.2016.01.038.

[19] M. Keskin and B. Çatay, "A matheuristic method for the electric vehicle routing problem with time windows and fast chargers," *Computers & Operations Research*, vol. 100, pp. 172–188, 2018. DOI: 10.1016/j.cor.2018.07.018.

[20] S. Zhou, D. Zhang, W. Yuan, Z. Wang, L. Zhou, and M. G. Bell, "Pickup and delivery problem with electric vehicles and time windows considering queues," *Transportation Research Part C: Emerging Technologies*, vol. 167, p. 104 829, 2024. DOI: 10.1016/j.trc.2024.104829.

[21] M. Nazari, A. Oroojlooy, L. V. Snyder, and M. Takać, "Reinforcement learning for solving the vehicle routing problem," *arXiv preprint arXiv:1802.04240*, 2018.

[22] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[23] C. Chen, Z. He, and L. Sun, "A machine learning approach to travel time prediction in urban areas," *Transportation Research Part C: Emerging Technologies*, vol. 99, pp. 102–118, 2019.

[24] H. Xu, X. Li, H. Qiu, and L. Yang, "A deep learning approach for electric vehicle energy consumption prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, pp. 2146–2156, 2020.

[25] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in neural information processing systems*, 2015, pp. 2692–2700.

[26] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" In *International Conference on Learning Representations (ICLR)*, 2019.

[27] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[28] M. A. Nowak, Ö. Ergun, and C. C. White, "Pickup and delivery with split loads," *Transportation Science*, vol. 42, no. 1, pp. 32–43, 2008. DOI: 10.1287/trsc.1070.0207.

[29] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959. DOI: 10.1287/mnsc.6.1.80.

[30] P. Toth and D. Vigo, *Vehicle Routing: Problems, Methods, and Applications* (MOS-SIAM Series on Optimization). SIAM, 2014.

[31] J. K. Lenstra and A. H. Rinnooy Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981. DOI: 10.1002/net.3230110211.

[32] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *CP-98: Fourth International Conference on Principles and Practice of Constraint Programming*, ser. Lecture Notes in Computer Science, vol. 1520, 1998, pp. 417–431.

[33] D. Pisinger and S. Røpke, "Large neighborhood search," in *Handbook of Metaheuristics*, M. Gendreau, Ed., 2nd ed., Accessed: 2025-04-28, Springer, 2010, pp. 399–420. [Online]. Available: https://orbit.dtu.dk/en/publications/61a1b7ca-4bf7-4355-96ba-03fcdf021f8f.

[34] W3Schools, *DSA Dijkstra's Algorithm*, Accessed: 2025-05-23. [Online]. Available: https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php.

[35] T. Ghandriz, B. Jacobson, L. Laine, and J. Hellgren, "Impact of automated driving systems on road freight transport and electrified propulsion of heavy vehicles," *Transportation Research Part C: Emerging Technologies*, vol. 115, 2020. DOI: 10.1016/j.trc.2020.102610.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY