UNIVERSITY

Master's Degree in ICT4SS

Politecnico di Torino

Master's Degree Thesis

A Heuristic for the Electric Vehicle Routing Problem with Time Windows and Stochastic Waiting Times

Supervisors

Prof. Edoardo FADDA

Prof. Ornella PISACANE

Prof. Domenico POTENA

Prof. Maurizio BRUGLIERI

Candidate

Rares Alexandru BALAN

April 2024

Summary

Logistics is one of the greatest sources of pollution. For this reason, electric vehicles for pick up and deliveries are starting to be common. In the last mile setting, the management of these vehicles is not a big challenge since batteries have enough charge to cover the entire day of usage. Nevertheless, in the mid-haul setting, the distances to be covered in a single working day are usually more than the vehicle's capability over a single charge, therefore a recharging stop is required. This paves the way for the application of optimization methods.

In this context, we consider the Electric Vehicle Routing Problem for defining the planning of a series of pick-ups spatially distributed so that the vehicles may need a recharge at a station. The proposed solution methodology uses an Adaptive Large Neighborhood Search in which the first stage aims at destroying and repairing the solution using the expected waiting time at the stations. The second stage uses the realized waiting time which could lead to customers' demand being violated, whereby a recourse operation rectifies the broken solution by sending a new vehicle to the interested node.

This is where the proposed heuristic comes in, which aims at improving horizontally on the consolidated methodologies of the literature. More specifically it strives to give alternative operators to the recourse step, considering other realistic situations that the operations would find themselves in. The alternative recourse strategies provide an exchange of the customer needs between vehicles and the possibility of trying to recharge less at a station to still be able to serve without violating the customer's demand.

Acknowledgements

Thanks to the continued support of my supervisor E. FADDA, to my family for always supporting me and my girlfriend who gave me the motivation to conclude my academic journey

Table of Contents

List of Tables

List of Figures

Acronyms

AFV
Alternative Fuel Vehicle

EV
Electric Vehicle

SoC
State of Charge

ALNS
Adaptive Large Neighborhood Search

LNS
Large Neighborhood Search

VRP
Vehicle Routing Problem

EVRP
Electric Vehicle Routing Problem

OF
Objective Function

Chapter 1

Introduction

This thesis work is based and expanded upon the research study of [1, Keskin et al.].

Section 1.1 introduces the motivations behind this work giving an overview of the environmental context.

Section 1.2 introduces the problem under study, explaining the basic characteristics and the solution organization.

Section 1.3 explains the ALNS from a theoretical point of view.

Section 1.4 introduces the outline of the work and the contribution proposed in this thesis work.

1.1 Motivations

Developed countries are witnessing a growing demand to mitigate the environmental impact of transportation and to create cleaner city centers by reducing various emissions. Addressing the environmental concerns is crucial, given that the transportation sector, as highlighted in a study by Quadrelli et al. ([2]), contributed to 24% of total CO_2 emissions in 2004. This represents a notable increase from the 20% reported in 1971, indicating a concerning upward trend in emissions. Without effective countermeasures, there is a risk that CO_2 emissions will continue to escalate.

To assess the relationship between CO_2 emissions and logistics performance, Mariano et al. ([3]) proposed a composite index. This index serves as a valuable tool for evaluating both the environmental friendliness and functionality of the transportation sector. By providing insights into the extent to which the transport industry is environmentally sustainable, the index also offers a glimpse into the countries' investments in reducing the carbon footprint associated with transportation. Such multidimensional indices contribute to a comprehensive understanding of the interplay between environmental considerations and logistical efficiency in the context of transportation.

1.2 Problem Definition

The problem at hand involves designing a set of routes for identical Electric Vehicles (EVs), each capable of making stops at stations to recharge their batteries. These routes must fulfill the demands of known customers, each with specific demand quantities and time windows. Notably, if a vehicle arrives before the early time window, it must wait until the designated time to serve the customer. However, arrival after the late time is strictly prohibited. Additionally, the State of

Charge (SoC) throughout the route must remain non-negative, with the option to visit a station for recharging if necessary. The EVs commence their journeys from the depot opening and must return before the depot closure.

Upon reaching a station, a vehicle may encounter congestion, necessitating a waiting period for its turn to recharge. The waiting time is subject to a known distribution, but the actual time remains uncertain. This waiting time can result in violations of time windows for subsequent customers, prompting recourse actions to rectify the violation.

The problem is structured as a two-stage model. In the first stage, routes are determined and computed. In the second stage, the routes are simulated up to the station, where the random waiting time is realized. The first-stage solution is then adjusted through a recourse action. The overarching objective is to find a set of routes that minimize the cost of the first stage combined with the expected cost of the recourse action. The first-term cost encompasses the vehicle fixed cost, driver cost, and energy cost.

Three available recourse actions address violations. Firstly, if a customer's time window is breached, a new vehicle is dispatched from the depot to serve that customer. In the second scenario, the violated customer may be assigned to a vehicle already en route; otherwise, the first action is implemented. Lastly, in the third case, the vehicle attempts partial recharging to minimize time spent at the station. If this proves insufficient, one of the first two actions is applied accordingly.

1.3 Adaptive Large Neighborhood Search

The Adaptive Large Neighborhood Search (ALNS) represents a meta-heuristic approach applied to tackle combinatorial optimization problems within an optimization framework. The primary mechanism of the ALNS involves a continuous cycle of destroying and repairing the solution through operators, facilitating the exploration of extensive neighborhoods within the search space. This methodology is an evolution of the Large Neighborhood Search (LNS), with the ALNS introducing dynamic adjustments to the search strategy.

The ALNS differentiates itself by dynamically assigning scores and weights to each method used in the destruction and repair process. These scores and weights undergo changes at predefined intervals or after a certain number of iterations. This dynamic adaptation allows more effective methods to exert a greater influence on the solution, enhancing their impact. Despite this emphasis on effectiveness, the ALNS maintains an element of stochasticity in the solution, ensuring a randomized mixing of solutions. This stochastic mixing is crucial for avoiding local optima and promoting a more robust exploration of the solution space.

In essence, the ALNS combines the foundational principles of the Large Neighborhood Search with adaptive mechanisms, providing a flexible and dynamic approach to optimizing solutions in combinatorial optimization problems. The dynamic adjustment of search strategies based on the performance of individual methods contributes to the algorithm's adaptability and efficiency in finding near-optimal or optimal solutions in complex optimization environments.

1.4 Outline and Contribution

This thesis aims to build upon the groundwork laid by Keskin et al. in their previous work, as documented in [1]. The primary focus is on introducing alternative recourse methods to address challenges within the proposed framework. Specifically, the thesis explores the potential of exchanging orders among vehicles in transit and the feasibility of implementing partial recharging at designated stations.

The structure of this thesis unfolds as follows: Chapter 2 provides an updated and comprehensive literature review, delving into the nuances of the problem and tracing its evolution over time. Chapter 3 elucidates the methodology employed to derive the proposed solutions. Subsequently, Chapter 4 presents the computational results and draws comparisons between the initially adopted solution and the alternatives proposed in this thesis. Finally, Chapter 5 encapsulates a succinct summary of the undertaken work and outlines potential avenues for future improvements and enhancements that can be applied to the problem at hand.

Chapter 2

Literature review

This chapter discusses the existing problems and methodologies relevant to the work of this thesis. More specifically the expansion of the problem and the proposed solutions for each stage.

Section 2.1 discusses the VRP, the most basic version of this problem, and the proposed methodologies analyzed over the years.

Section 2.2 is an in-depth analysis of the EVRP and the proposed methodologies in the literature, relevant to this study.

Section 2.3 discusses the stochastic EVRP and what it means from a research point of view.

Section 2.4 presents the solution and methodology proposed in the work of Keskin et al.[1]

2.1 Vehicle Routing Problem and solution methodologies

Research on the Vehicle Routing Problem (VRP) has been ongoing since the 1960s, as documented in [4]. The fundamental objective of the VRP is to optimize the determination of an optimal set of routes for vehicles tasked with distributing or picking up goods. Mathematically, the VRP falls into the NP-hard class of problems, indicating its computational complexity. While exact solution methods exist, they are often time-consuming and impractical for real-world applications. Consequently, the field has witnessed the development of approximation methods, including heuristics and meta-heuristics.

Various techniques have been devised to address the VRP, among which are heuristics and meta-heuristics due to their ability to provide reasonably good solutions in a more computationally feasible manner. Noteworthy examples include simulated annealing [5], tabu search [6], savings algorithms [7], I1 route construction method [8], and the 2-opt optimization algorithm [9].

Simulated annealing involves adjusting the initial solution and determining whether to accept the modification based on a temperature parameter that allows for the acceptance of inferior solutions. As the algorithm advances, the temperature decreases, gradually steering the objective function towards a near-optimal value.

Tabu search revolves around refining the solution by referencing a list of recently visited solutions, which helps prevent the algorithm from becoming trapped in local minima. This method may involve restricting certain actions temporarily to encourage exploration of different solution options.

The savings algorithm systematically combines pairs of routes into single, more efficient routes by identifying and exploiting opportunities for consolidation. This process repeats iteratively until no further significant savings can be achieved, resulting in a set of optimized routes that minimize the total distance traveled by vehicles.

The I1 construction method is primarily employed to generate initial solutions. It begins by selecting a specific customer as the starting point and then progressively adds additional customers to the route. This selection process is guided by factors such as distance, time, or demand.

The 2-opt algorithm operates by iteratively swapping pairs of edges in a given route and evaluating whether the resulting route is shorter. If the new route is shorter, the edge swap is accepted, otherwise it is rejected. This process continues iteratively until no further improvements can be made.

These methodologies have been employed to effectively solve the VRP, offering practical solutions to a problem that holds significance in logistics and transportation optimization.

2.2 The Electric Vehicle Routing Problem and solution methodologies

The research conducted by Erdelic et al. [10] provides a comprehensive analysis of the existing body of work on the Electric Vehicle Routing Problem (EVRP) and its solutions. Within this broader context, several notable studies have delved into specific facets of the EVRP, contributing valuable insights to the field.

Juan et al. [11] conduct a thorough examination of the environmental, strategic, and operational challenges associated with Electric Vehicles (EVs) in logistics. They specifically address the impact of electric vehicles on carbon emissions, exploring various components of the problem. Their focus encompasses battery swap technology, optimal charging station locations, constraints on the number of charges, and the intricacies of the charging network.

Similarly, Margaritis et al. [12] center their attention on the development of batteries, charger compatibility, systematic energy management, the utilization of cooling and heating systems, energy sources, novel policies, and the absence of optimization procedures to minimize EV routing and scheduling decisions.

Pelletier et al. [13] contribute to the technical background of Electric Vehicle types and batteries, EV market penetration, and competitiveness of EVs, and provide an overview of the existing research landscape in the domain of Electric Vehicles in transportation.

The uncertain waiting time at public recharging stations has attracted the attention of multiple researchers. Sweda et al. [14] delve into the problem of computing paths between nodes, with a specific focus on station availability along these paths. On the other hand, Kullman et al. [15] address the EVRP by considering potential recharge points at public recharging stations or the depot, explicitly accounting for waiting times. Their approach involves the development of routing policies that anticipate queue dynamics, wherein an arriving EV may either wait in a queue or decide to continue its route without recharging, subsequently replanning the remainder of its journey. These studies collectively contribute to a nuanced understanding of the EVRP and its various dimensions.

2.3 The Stochastic Electric Vehicle Routing Problem and Solution Methodologies

Stochastic Vehicle Routing Problems (SVRPs) involve routing fleets where certain parameters are not predetermined. In the specific SVRP presented in [1], the variability lies in the stochastic nature of service times, meaning the time required to serve each customer is uncertain.

To address SVRPs with stochastic service times, two primary methodologies are commonly employed: chance-constrained and multi-stage stochastic solutions. In chance-constrained approaches, the aim is to ensure constraints are met with a specified probability. However, if constraints are not satisfied, recourse actions are typically lacking. On the other hand, multi-stage stochastic methodologies compute an initial solution without knowledge of the actual stochastic parameter realization. After the stochastic parameters are revealed, the solution is re-optimized through recourse actions.

Recourse actions involve adjusting the initial solution based on the actual realization of stochastic parameters. Proposed actions include permitting late arrivals at customers or the depot ([16]) and skipping certain customers ([17]). Both exact and meta-heuristic solution methods have been developed for SVRPs. These methods may derive probability distributions through analytical approaches or experimental simulations.

For a comprehensive exploration of SVRPs, including various stochastic parameters, [18] provides an in-depth overview, covering different formulations, solution methods, and applications in the literature. In summary, SVRPs demand sophisticated modeling and solution strategies to navigate uncertainty and make robust decisions in the face of unpredictable parameters.

2.4 Keskin solution

Keskin et al. [1] introduce a simulation-based Adaptive Large Neighborhood Search (ALNS) heuristic for the Electric Vehicle Routing Problem (EVRP). The ALNS begins by constructing the first-stage solution using expected values for waiting times. This initial solution is then subjected to a simulator, evaluating the consequences of the vehicle adhering to the proposed routing. At each station, the waiting time is realized, and based on this, the second-stage solution is formulated. The total cost of the recourse action is computed, considering multiple scenarios.

The ALNS is designed with two distinct classes of operators: destroy operators and repair operators. In this thesis, these methods are not only mirrored but also expanded upon, providing a detailed exploration and extension of the proposed techniques. This extension involves a comprehensive explanation of the specific destroy and repair operators employed in the ALNS.

Moving to the second stage, the problem is built upon the output of the ALNS first stage, given that it provides a feasible solution. However, if the station within the route causes a delay significant enough to compromise this feasibility, a corrective action is triggered. In such cases, a new vehicle is dispatched from the depot to fulfill the needs of the violated customers, ensuring that service commitments are met.

This iterative process is repeated for a predetermined number of iterations, and at each step, a new solution is determined. The decision-making process for selecting a new solution is based on improvements to the objective function or the acceptance of denied solutions, employing a simulated annealing criterion. This strategy aids in better exploring the solution space, allowing the algorithm to adapt and refine its solutions throughout multiple iterations. The intricate interplay between the first and second stages, coupled with the dynamic adjustment of the solution strategy, forms the core of the proposed simulation-based ALNS heuristic for the EVRP.

Chapter 3

Methodology

In this chapter, the problem methodology is explained and analyzed in-depth.

Section 3.1 introduces the structure of the instances and the setup of the problem in general.

Section 3.2 is an in-depth look at the implementation of the core algorithm of this thesis work, the ALNS, explaining how every operator works and how the scores and weights are updated.

Section 3.3 introduces and explains the recourse actions used in the second stage of the problem to get the expected recourse cost.

Finally, Section 3.4 discusses the simulated annealing and how it impacts the final solution.

3.1 Instances and problem setup

The instances examined in this study are derived from the Solomon dataset [19], augmented with the specific characteristics related to electric vehicles. The dataset used can be accessed via the link provided in the paper by Desaulniers et al. [20]. Each instance belongs to one of three types: C, characterized by a clustered arrangement of elements based on geographical logic; R, where the positioning is entirely random; and RC, which combines both randomly and clustered positioned customers.

The problem-solving approach is structured into five main steps and draws inspiration from the heuristic proposed by Keskin et al. [1]:

The algorithm begins by constructing a base solution through a greedy approach. This involves searching for the nearest reachable customer, serving it, and continuing until the vehicle's return to the depot is imminent. This initial step often results in a solution with a relatively high cost, laying the groundwork for subsequent refinement.

The Adaptive Large Neighborhood Search (ALNS) cycle commences with this initial solution as its foundation. The first stage of the ALNS involves a process of destroying and repairing the solution. Both stations and customers undergo modifications, with the distinction that station modifications occur only every five iterations. After this step potential empty routes or routes with only one customer are eliminated, effectively reducing overall costs. Importantly, during this stage, there is no explicit check for solution feasibility.

Following the first stage, the algorithm verifies the feasibility of the solution. In cases where the solution becomes infeasible, the applied methods receive low scores, diminishing their probability of being selected. However, if the solution remains feasible, the algorithm proceeds to compute the expected recourse cost in the second stage, considering various recourse application heuristics.

Once the expected recourse cost is computed, the algorithm evaluates whether to accept the solution. If the overall cost is lower than the best cost encountered thus far, the solution

is accepted outright. Alternatively, the solution may be accepted with simulated annealing, enhancing the statistical mixing of solutions. If both tests fail, indicating that the solution is not optimal, the proposed solution is rejected, and the previous one is reinstated.

Periodically, every ten iterations, the scores of the first-stage operators are assessed, and their weights are updated based on performance. The algorithm then proceeds to the next batch of iterations, resetting the scores to zero. This iterative process ensures that the algorithm dynamically adapts its strategy over multiple iterations, balancing exploration and exploitation in the search for an optimal solution.

3.2 ALNS implementation

The ALNS is implemented based on the organization in [1, Keskin et al.]'s paper.

3.2.1 ALNS destroy operators

The destroy operators modify the given solution by removing a certain number of elements from it. In this project, there are operators for both the customers and the stations.

Customer destroy operators

In the context of this thesis, the term "Customer Destroy" refers to a family of operators designed to remove a certain number of customers from the current solution (γ_c). These operators are characterized by distinct criteria, and once a customer is removed, it is temporarily saved and passed on to the repair operator for potential inclusion in the revised solution.

This thesis utilizes 11 variations of destruction operators, primarily influenced by the work of Keskin et al. [1] and their recommended implementation. These operators are as follows: random customer removal, worst distance destroy, worst time destroy, Shaw destroy, demand-based destroy, time-based destroy, proximity-based destroy, random route destroy, zone destroy, greedy route removal, and probabilistic worst removal.

Random customer destroy

The "Random Customer Removal" operator, as its name suggests, involves the selection of a customer from the available list of elements and subsequently removing that customer from the current solution. The pseudocode for the implementation of this operator is outlined in Algorithm 1.

Worst distance customer destroy

The "Worst Distance Removal" procedure involves ranking the customers in decreasing order based on the sum of the distance costs covered from a node to both its successor and predecessor. After this ranking, the procedure generates a random number between 0 and 1. A parameter $\kappa \geq 1$, referred to as the determinism factor, serves as a stochasticity index. If κ is closer to 1, the choice of which customer to remove becomes more random.

In simpler terms, the worst distance removal method identifies customers that, when removed, would result in the most significant increase in the overall distance cost. The stochasticity introduced by the determinism factor ensures variability in the customer selection process. A higher κ makes the choice more random, while a lower value leans towards a more deterministic selection.

Algorithm 1 Random customer removal pseudo-code

```
1: procedure RANDOM DESTROY CUSTOMER(solution)
2:     ▷ solution contains all the information on routes and vehicles
3:     c_r ← empty list                                  ▷ customers to remove list
4:     while length(c_r) < γ_c do
5:         c ←random customer to remove
6:         if c ∉ c_r then
7:             append c to c_r
8:         end if
9:     end while
10:    Remove customers from solution
11:    return c_r                                        ▷ Return removed customers
12: end procedure
```

Algorithm 2 provides the pseudocode that outlines the step-by-step implementation of the worst distance removal procedure within the algorithm.

Algorithm 2 Worst distance removal pseudo-code

```
1: procedure WORST DISTANCE DESTROY CUSTOMER(solution)
2:     ▷ solution contains all the information on routes and vehicles
3:     c_r ← empty list                                  ▷ customers to remove list
4:     d_c ← empty list                                  ▷ distance costs list
5:     for route ∈ routes do
6:         for node ∈ route do
7:             cost_arc1 ← cost from node to predecessor
8:             cost_arc2 ← cost from node to successor
9:             append |cost_arc1 + cost_arc2| to d_c
10:        end for
11:    end for
12:    while length(c_r) < γ_c do
13:        n ←random value ∈ [0,1)
14:        pos ← ⌊n^κ * length(d_c)⌋
15:        c ← customer at position pos
16:        if c ∉ c_r then
17:            append c to c_r
18:        end if
19:    end while
20:    Remove customers from solution
21:    return c_r                                        ▷ Return removed customers
22: end procedure
```

Worst time customer destroy

The "Worst Time Destroy" procedure operates like Algorithm 2. However, instead of focusing on the cost of the arc, it assesses the difference in time between the early time window of the customer and the arrival of the vehicle to the element. The pseudocode implementation is provided in Algorithm 3.

In essence, this procedure identifies customers whose removal would result in the most significant negative impact on the time-related constraints. It evaluates the difference between the early time window of a customer and the arrival time of the vehicle at that customer. By ranking customers based on this difference, the algorithm aims to select customers that, when removed, would disrupt the existing schedule the most.

This method provides a complementary approach to solution refinement, specifically targeting time-related constraints. The pseudocode offers a detailed representation of how the worst-time destroy procedure is implemented within the algorithm.

Shaw customer destroy

The "Shaw Customer Destroy" procedure introduces a measure of relatedness between customers, aiming to remove the most closely related customers to a particular node. The relatedness is calculated using the following formula:

$$\Gamma_{ij} = \lambda c_{ij} + \mu|c_{ei} - c_{ej}| + \phi|\Omega_{ij}| + \nu|\delta_i - \delta_j|$$

Algorithm 3 Worst time removal pseudo-code

```
1: procedure WORST TIME DESTROY CUSTOMER(solution)
2:     ▷ solution contains all the information on routes and vehicles
3:     c_r ← empty list                                   ▷ customers to remove list
4:     t_c ← empty list                                   ▷ time costs list
5:     for route ∈ routes do
6:         for node ∈ route do
7:             time ← |arrival − c_e|                     ▷ c_e is the early customer time window
8:             append time to d_c
9:         end for
10:    end for
11:    while length(c_r) < γ_c do
12:        n ←random value ∈ [0,1)
13:        pos ← ⌊n^κ ∗ length(d_c)⌋
14:        c ← customer at position pos
15:        if c ∉ c_r then
16:            append c to c_r
17:        end if
18:    end while
19:    Remove customers from solution
20:    return c_r                                         ▷ Return removed customers
21: end procedure
```

Here:

- λ is the Shaw factor that relates to the distance between two customers.
- μ relates to the difference in the early time window for customers.
- ϕ identifies whether two customers are on the same route.
- ν relates to the difference in demand between two customers.

The relatedness values are then arranged in decreasing order, and a position is chosen like the selection process in Algorithms 2 and 3. Algorithm 4 provides the pseudocode implementation which offers a step-by-step representation of how the Shaw customer destroy procedure is implemented within the algorithm, showcasing the incorporation of various relatedness factors for intelligent customer removal.

In essence, the Shaw removal procedure assesses the inter-customer relationships based on multiple factors, including spatial distance, temporal constraints, route similarity, and demand differences. By removing customers with higher relatedness values, the algorithm aims to disrupt clusters of closely related customers, potentially leading to more effective solution improvements.

Demand, time, proximity customer destroy

The demand-based, time-based, and proximity-based removal procedures are all built on Algorithm 4. The only difference in the methods is that all other parameters are set to 0, except the one under study. In the demand based, λ, ϕ, μ parameters are all 0 while ν is set to ν_0. For the time based only μ is set to μ_0 while in the proximity-based only λ is set to λ_0. To avoid redundancy the pseudo-code of the procedures is omitted since they are equal to Algorithm 4.

The demand-based, time-based, and proximity-based removal procedures are variations of the Shaw removal procedure (Algorithm 4). The distinction lies in the setting of parameters in these methods, with all parameters other than the one under study being set to 0.

- Demand-Based Removal: In this procedure, λ, ϕ, and μ are all set to 0, while ν is assigned a value (ν_0). This means that only the demand-related parameter ν influences the relatedness measure.
- Time-Based Removal: This procedure sets μ to a value (μ_0) while all other parameters (λ, ϕ, ν) are set to 0. Thus, only the time-related parameter μ affects the computation of relatedness.
- Proximity-Based Removal: In this procedure, λ is set to a value (λ_0), while all other parameters (ϕ, μ, ν) are set to 0. Consequently, only the spatial distance-related parameter λ plays a role in determining relatedness.

To avoid redundancy, the pseudocode for these procedures is omitted since they share similarities with Algorithm 4.

Random route customer destroy

The "Random Route Customer Destroy" procedure is straightforward in its approach. It involves randomly selecting a route, then removing up to γ_c customers from that route.

Algorithm 5 provides the pseudocode for a step-by-step representation of how the random route customer destroy procedure is implemented within the algorithm.

Algorithm 4 Shaw removal pseudo-code

procedure SHAW DESTROY CUSTOMER(solution)
  ▷ solution contains all the information on routes and vehicles
  $c_r \leftarrow$ empty list ▷ customers to remove list
  $\Gamma_i \leftarrow$ empty list ▷ Relatedness to node i list
  $\lambda \leftarrow \lambda_0$ ▷ λ_0 is the initialized parameter
  $\mu \leftarrow \mu_0$ ▷ μ_0 is the initialized parameter
  $\phi \leftarrow \phi_0$ ▷ ϕ_0 is the initialized parameter
  $\nu \leftarrow \nu_0$ ▷ ν_0 is the initialized parameter
  $c_i \leftarrow$ random customer
  for $c_j \in$ customers **do**
    $\Gamma_{ij} \leftarrow \Gamma_{ij} + distance_{ij} \times \lambda$
    $\Gamma_{ij} \leftarrow \Gamma_{ij} + |c_{ei} - c_{ej}| \times \mu$ ▷ c_e is the early customer time window
    $\Gamma_{ij} \leftarrow \Gamma_{ij} + |\delta_i - \delta_j| \times \nu$ ▷ δ is the customer demand
    for route ∈ routes **do**
      if c_i and c_j ∈route **then**
        $\Omega_{ij} \leftarrow -1$
      end if
    end for
    $\Gamma_{ij} \leftarrow \Gamma_{ij} + \Omega_{ij} \times \phi$
    append Γ_{ij} to Γ_i
  end for
  while length(c_r) $< \gamma_c$ **do**
    $n \leftarrow$random value ∈ [0,1)
    $pos \leftarrow \lfloor n^{\kappa} * length(\Gamma_i) \rfloor$
    $c \leftarrow$ customer at position pos
    if $c \notin c_r$ **then**
      append c to c_r
    end if
  end while
  Remove customers from solution
  return c_r ▷ Return removed customers
end procedure

Algorithm 5 Random route customer removal pseudo-code

procedure RANDOM ROUTE DESTROY CUSTOMER(solution)
  ▷ solution contains all the information on routes and vehicles
  $c_r \leftarrow$ empty list ▷ customers to remove list
  $j \leftarrow$ random route
  while length(c_r) $< \gamma_c$ **do**
    $c \leftarrow$customer in route j to remove
    if $c \notin c_r$ **then**
      append c to c_r
    end if
  end while
  Remove customers from solution
  return c_r ▷ Return removed customers
end procedure

Zone customer destroy

The "Zone Removal" procedure divides the space occupied by customers into four zones. It then removes up to γ_c customers from a specific zone. Algorithm 6 provides the pseudocode for the implementation.

In simpler terms, the space is spatially divided into distinct zones, and the algorithm systematically removes customers from each zone until the desired number of removals is achieved. This procedure helps in exploring the impact of removing customers from different spatial regions, contributing to the diversity in the solution search space.

Algorithm 6 Zone customer removal pseudo-code

```
1:  procedure ZONE DESTROY CUSTOMER(solution)
2:      ▷ solution contains all the information on routes and vehicles
3:      c_r ← empty list                                  ▷ customers to remove list
4:      j ← random zone
5:      while length(c_r) < γ_c do
6:          c ←customer in zone j to remove
7:          if c ∉ c_r then
8:              append c to c_r
9:          end if
10:     end while
11:     Remove customers from solution
12:     return c_r                                        ▷ Return removed customers
13: end procedure
```

Greedy customer destroy

The "Greedy Route Removal" procedure involves the computation of the cost for each route in the solution. Subsequently, the routes are ranked in decreasing order based on their costs, and up to γ_c customers are removed from the top-ranked route. Algorithm 7 provides the pseudocode for the implementation.

In simpler terms, this procedure identifies the route with the highest cost, likely indicating inefficiency, and selectively removes customers from that route. By focusing on the most costly route, the algorithm aims to strategically improve overall solution quality.

Probabilistic customer destroy

The "Probabilistic Worst Removal" procedure focuses on removing customers that might lead to a violation of their successor's time window. Since waiting time is only realized at charging stations, the probability of violating time windows is considered for all successive customers. The procedure ranks customers based on this probability and then removes the first γ_c customers from the ranked list. Algorithm 8 provides the pseudocode for the implementation.

In simpler terms, this procedure identifies and removes customers whose presence in the solution could potentially cause violations of time windows for their successors. By considering the waiting time realization at charging stations, it calculates the probability of time window violations for successive customers and prioritizes their removal.

Algorithm 7 Greedy route customer removal pseudo-code

```
1: procedure GREEDY ROUTE DESTROY CUSTOMER(solution)
2:     ▷ solution contains all the information on routes and vehicles
3:     c_r ← empty list                                   ▷ customers to remove list
4:     cost ← empty list                                  ▷ route costs list
5:     for route ∈ routes do
6:         cost_route ← cost of the route
7:         append cost_route to cost
8:     end for
9:     j ← most expensive route in cost
10:    while length(c_r) < γ_c do
11:        c ←customer in j to remove
12:        if c ∉ c_r then
13:            append c to c_r
14:        end if
15:    end while
16:    Remove customers from solution
17:    return c_r                                         ▷ Return removed customers
18: end procedure
```

Algorithm 8 Probabilistic worst customer removal pseudo-code

```
1: procedure PROBABILISTIC WORST DESTROY CUSTOMER(solution)
2:     ▷ solution contains all the information on routes and vehicles
3:     c_r ← empty list                                   ▷ customers to remove list
4:     p_inf ← empty list                                 ▷ probability of infeasibility list
5:     for route ∈ routes do
6:         for node ∈ route do
7:             prob ← 0
8:             for n ∈ scenarios do
9:                 if node after station and vehicle arrival > c_enode then
10:                    prob ← prob + 1
11:                end if
12:            end for
13:            prob ← prob/scenarios
14:
15:            append prob_route to p_inf
16:        end for
17:    end for
18:    while length(c_r) < γ_c do
19:        c ← most probable customer
20:        if c ∉ c_r then
21:            append c to c_r
22:        end if
23:    end while
24:    Remove customers from solution
25:    return c_r                                         ▷ Return removed customers
26: end procedure
```

Station Destroy

The family of "Station Destroy" operators involves removing a certain number of stations (γ_s) from the current solution, guided by various criteria. Once the stations are removed, the modified solution is passed on to the repair operator for adjustment.

In this thesis, two specific station removal operators are employed: the "Random Destroy" and the "Longest Waiting Time Destroy."

Random station destroy

The "Random Station Removal" procedure, as its name suggests, involves randomly selecting a station from the current solution and removing it. Algorithm 9 provides the pseudocode for the implementation.

In simpler terms, this procedure introduces a stochastic element by randomly choosing a station and excluding it from the current solution. The randomness contributes to the exploration of various possibilities for station removal during the optimization process.

Algorithm 9 Random station removal pseudo-code

```
1:  procedure RANDOM DESTROY STATION(solution)
2:      ▷ solution contains all the information on routes and vehicles
3:      s_r ← empty list                                  ▷ stations to remove list
4:      while length(s_r) < γ_s do
5:          s ←random station to remove
6:          if s ∉ s_r then
7:              append s to s_r
8:          end if
9:      end while
10:     Remove station from solution
11:     return s_r                                        ▷ Return removed stations
12: end procedure
```

Longest waiting time station destroy

The "Longest Waiting Time Station Removal" procedure focuses on removing charging stations that contribute the most to waiting time. The objective is to allow the repair operator to select a more convenient station for the vehicles. Algorithm 10 provides the pseudocode for the implementation.

In simpler terms, this procedure identifies charging stations based on the waiting time they cause and prioritizes the removal of stations that result in the most extended waiting times for vehicles. By targeting stations associated with higher waiting times, the algorithm aims to improve the overall efficiency of the solution.

3.2.2 ALNS repair operators

The "Repair Operators" play a crucial role in modifying the given solution by reintegrating the elements that were previously removed. The goal is to enhance the existing solution by strategically placing the removed elements in different positions.

Algorithm 10 Longest waiting time station removal pseudo-code

```
1: procedure Longest Waiting Time Destroy Station(solution)
2:     ▷ solution contains all the information on routes and vehicles
3:     s_r ← empty list                                  ▷ stations to remove list
4:     while length(s_r) < γ_s do
5:         E_tw ← empty list
6:         for station in stations do
7:             E_twi ← expected waiting time of station i
8:             append E_twi to E_tw
9:         end for
10:        s_e ← E_tw sorted in increasing order
11:        for s ∈ s_e do
12:            if s ∉ s_r then
13:                append s to s_r
14:            end if
15:        end for
16:    end while
17:    Remove station from solution
18:    return s_r                                        ▷ Return removed stations
19: end procedure
```

In the case of "Customer Repair," the removed customers are reinserted into the solution, potentially in a different spot on the same route or even on an entirely different route. This process allows for the exploration of different customer placements to improve the overall solution.

On the other hand, in "Station Repair," a removed station is reintroduced to the route. It's important to note that the station added during repair may not necessarily be the same as the one that was removed. This flexibility allows for a dynamic adjustment of charging station locations to optimize the solution.

It's worth mentioning that the repair operators' step in the Adaptive Large Neighborhood Search (ALNS) does not guarantee that the repaired solution will be feasible. The algorithm explores various possibilities during repair, and feasibility is ensured in subsequent steps or iterations.

Customer repair

The family of "Customer Repair" operators focuses on reintegrating previously removed customers into the solution, intending to enhance its overall quality. In this thesis, four specific repair operators are employed: the "Greedy Repair", the "Naive Greedy Repair", the "Probabilistic Greedy Repair", and the "Probabilistic Greedy with Confidence Repair".

Greedy customer repair

The "Greedy Customer Repair" procedure aims to reintegrate previously removed customers into the solution. The approach taken is to explore every possible position for each customer on the routes and select the spot that minimally increases the cost of the solution, recalculating this cost after every customer to find its best position. This process is repeated for every γ_c customer that was removed during the destruction phase. Algorithm 11 provides the pseudocode for the implementation.

In simpler terms, for each removed customer, the algorithm systematically evaluates the impact of placing that customer at various positions within the routes. The goal is to identify the placement that results in the smallest increase in the overall solution cost. By prioritizing positions that least impact the cost, the algorithm seeks to optimize the reintroduction of customers into the solution.

Naive Greedy customer repair

The "Naive Greedy Customer Repair" procedure aims to reintegrate previously removed customers into the solution. The approach taken is to explore every possible position for each customer on the routes and select the spot that minimally increases the cost of the solution. This process is repeated for every γ_c customer that was removed during the destruction phase. To avoid redundancy, the implementation is omitted so refer to Algorithm 11.

Probabilistic greedy customer repair

The "Probabilistic Greedy Customer Repair" approach shares similarities with the greedy repair method in that it attempts to insert a customer at every possible position within each route. However, this approach introduces a probabilistic element, incorporating the expected recourse cost into the ranking of potential positions. The algorithm considers not only the immediate impact on the solution cost but also the overall expected cost, choosing positions that minimize the expected impact.

Algorithm 12 provides the pseudocode for the implementation of this procedure.

Algorithm 11 Greedy customer repair pseudo-code

```
1: procedure GREEDY REPAIR CUSTOMER(solution, removed_elements)
2:     ▷ solution contains all the information on routes and vehicles
3:     ▷ removed_elements contains all the elements removed at the destroy step
4:     for customer in removed_elements do
5:         max_cost_difference ← − inf
6:         for route in routes do
7:             for spot in route do
8:                 add customer in spot at route
9:                 cost_difference ← cost difference of insertion
10:                if cost_difference > max_cost_difference then
11:                    best_spot ← spot
12:                    best_route ← route
13:                end if
14:            end for
15:        end for
16:        add customer in best_spot of route best_route
17:    end for
18:    return solution
19: end procedure
```

In simpler terms, for each removed customer, the algorithm evaluates potential positions within the routes. The choice of position is not solely based on immediate cost considerations but also factors in the expected recourse cost. By probabilistically selecting positions with lower overall expected costs, the algorithm takes a more nuanced approach to customer reintegration, balancing immediate and potential future impacts on the solution.

Algorithm 12 Probabilistic greedy customer repair pseudo-code

```
1: procedure PROBABILISTIC GREEDY REPAIR CUSTOMER(solution, removed_elements)
2:     ▷ solution contains all the information on routes and vehicles
3:     ▷ removed_elements contains all the elements removed at the destroy step
4:     for customer in removed_elements do
5:         max_cost_difference ← −inf
6:         for route in routes do
7:             for spot in route do
8:                 add customer in spot at route
9:                 cost_difference ← cost difference of insertion + expected recourse cost
10:                if cost_difference > max_cost_difference then
11:                    best_spot ← spot
12:                    best_route ← route
13:                end if
14:            end for
15:        end for
16:        add customer in best_spot of route best_route
17:    end for
18:    return solution
19: end procedure
```

Probabilistic greedy customer repair with confidence

The "Probabilistic Greedy Customer Repair with Confidence" operates similarly to the greedy customer repair approach, attempting to insert a customer at every possible position within each route. However, it introduces an additional layer of complexity by computing the probability that the route remains feasible after the repair. If this probability falls outside a specified confidence interval, the insertion is discarded. The algorithm then selects the position with the lowest insertion cost, and the customer is added to the solution.

Algorithm 13 provides the pseudocode for the implementation of this procedure.

In simpler terms, for each removed customer, the algorithm assesses potential positions within the routes. The decision to insert the customer considers not only the immediate cost implications but also the feasibility of the route after the repair. If the computed probability of feasibility falls within a certain confidence interval, the insertion is accepted. Otherwise, it is discarded, and the algorithm seeks an alternative position. The final choice is based on minimizing the overall insertion cost.

Station repair

The family of "Station Repair" operators focuses on integrating stations into the solution, which does not necessarily include the stations removed in the destroy step, intending to enhance its

Algorithm 13 Probabilistic greedy customer repair with confidence pseudo-code

```
1: procedure PROBABILISTIC GREEDY CONFIDENCE REPAIR CUSTOMER(solution, removed_elements)
2:     ▷ solution contains all the information on routes and vehicles
3:     ▷ removed_elements contains all the elements removed at the destroy step
4:     for customer in removed_elements do
5:         max_cost_difference ← −inf
6:         for route in routes do
7:             for spot in route do
8:                 add customer in spot at route
9:                 cost_difference ← cost difference of insertion
10:                p_feasible ← probability that route is feasible
11:                if p_feasible > greedy_confidence then
12:                    if cost_difference > max_cost_difference then
13:                        best_spot ← spot
14:                        best_route ← route
15:                    end if
16:                else
17:                    discard insertion
18:                end if
19:            end for
20:        end for
21:        add customer in best_spot of route best_route
22:    end for
23:    return solution
24: end procedure
```

overall quality. In this thesis, two specific repair operators are employed: the "Deterministic Best Repair", and the "Probabilistic Best Repair".

Deterministic best station repair

The "Deterministic Best Station Repair" procedure focuses on reintegrating stations into the solution at previously removed spots. The approach involves exploring every possible station that was removed and assessing the maximum cost difference between the old station and the new one. Subsequently, the procedure inserts the station that yields the best cost improvement back into the solution.

Algorithm 14 provides the pseudocode for the implementation of this procedure.

In simpler terms, for each removed station, the algorithm evaluates various possibilities of reintegrating stations into the solution. The goal is to identify the station that, when inserted, results in the maximum improvement in the overall solution cost. By systematically exploring these possibilities, the algorithm optimizes the reintroduction of stations into the solution.

Algorithm 14 Probabilistic greedy customer repair with confidence pseudo-code

```
1: procedure DETERMINISTIC BEST STATION REPAIR(solution, removed_elements)
2:     ▷ solution contains all the information on routes and vehicles
3:     ▷ removed_elements contains all the elements removed at the destroy step
4:     c_l ← empty list
5:     ▷ list of base costs
6:     for route in solution do
7:         append cost_solution to c_l
8:     end for
9:     for route in routes do
10:        for spot in route do
11:            insertion_c ← empty list
12:            add station in spot at route
13:            append insertion_cost − c_l to insertion_c
14:            if insertion_c > max_cost_difference then
15:                insert station at the spot
16:            end if
17:        end for
18:    end for
19:    return solution
20: end procedure
```

Probabilistic Best Station Repair

The "Probabilistic Best Station Repair" procedure shares similarities with the "Deterministic Best Station Repair." It also focuses on reintegrating stations into the solution at previously removed spots. The primary distinction lies in the consideration of both the minimal insertion cost and the cost of the recourse stage. This dual consideration adds a probabilistic element to the decision-making process, incorporating the expected cost implications of station reintroduction.

While the pseudocode for the "Probabilistic Best Station Repair" is omitted, it operates like Algorithm 14. For each removed station, the algorithm assesses various possibilities of reintegrating stations into the solution, aiming to identify the station that, when inserted, results in the optimal

balance between minimal insertion cost and expected recourse cost. This probabilistic approach adds complexity to the decision-making process, considering both immediate and potential future impacts on the solution.

3.2.3 Operators score and weight

The operators mentioned earlier work collaboratively within the Adaptive Large Neighborhood Search (ALNS) framework to iteratively refine the solution. The ALNS employs a dynamic scoring mechanism and weight adjustment for each operator to adapt and improve the search strategy over iterations.

The ALNS assigns scores to operators based on their impact on the current solution state.

- If the operators result in an infeasible solution, the score is set to 0.
- If the solution is rejected based on the new cost, the simulated annealing attempts to accept the new solution, and a score of 1 is assigned if successful.
- If the solution improves the cost but is not the best solution yet, the score is set to 3.
- If the solution provides the best cost as of yet, the score is 5.

After every N iterations, the weights of both customer operators and station operators are updated based on the total score accumulated over these iterations. The weights are adjusted to reflect the performance of each operator, giving higher weight to operators that have consistently led to improved solutions. In essence, the ALNS dynamically adjusts the influence of each operator based on their historical performance. By assigning scores and updating weights, the ALNS strives to increase the likelihood of choosing operators that have demonstrated effectiveness in improving the solution, while gradually diminishing the influence of less effective operators. This adaptive approach contributes to the algorithm's ability to explore the solution space efficiently and converge towards optimal or near-optimal solutions.

3.3 Recourse actions

Following the completion of the first stage, which involves the application of Adaptive Large Neighborhood Search (ALNS), the algorithm proceeds to the second stage, which focuses on computing the expected cost of recourse actions. In this thesis, three recourse actions are proposed: the basic recourse mirroring the method proposed by Keskin et al. [1], the pickup exchange recourse, and the partial recharge recourse.

3.3.1 Basic recourse

The "Basic Recourse" procedure closely follows the approach proposed by Keskin et al. [1]. The method involves the realization of waiting times at the stations, followed by a thorough check for feasibility concerning all subsequent customers. If any customer is found to violate the specified time window, a straightforward corrective action is taken: a new vehicle is dispatched from the depot to serve the customers that were identified as violating their time window.

3.3.2 Pickup exchange recourse

In the "Pickup Exchange Recourse" procedure, the algorithm responds to the realization of waiting times at stations by conducting a feasibility check on all subsequent customers. If it identifies a customer violating their specified time window, the algorithm initiates a corrective action. Specifically, it attempts to modify the route of another vehicle by incorporating the problematic customer into its itinerary. The aim is to address the violation and ensure the overall feasibility of the routes by redistributing the responsibility for serving customers among different vehicles.

3.3.3 Partial recharge recourse

In the "Partial Recharge Recourse" procedure, the algorithm responds to the realization of waiting times at stations by conducting a feasibility check on all subsequent customers. If it identifies a customer violating their specified time window, the algorithm initiates a corrective action. In this case, the vehicle attempts to limit its charging process by interrupting it prematurely. The goal is to address the violation and enable the vehicle to serve the customers without any time window violations. This recourse action focuses on adjusting the charging strategy dynamically to ensure that the overall feasibility of the routes is maintained, particularly in situations where violations are detected.

3.4 Simulated annealing

The Simulated Annealing approach in this work serves the purpose of accepting a rejected solution during the main iteration loop, enabling exploration of the search space and avoiding getting trapped in local minima. The mechanism involves generating a random number uniformly distributed between 0 and 1. This random number is then compared against the value of an exponential distribution, which is determined by the difference in the objective function between the current solution and the previously rejected solution according to the following formula:

$$e^{\frac{(OF_{previous} - OF_{current})}{k * T}}$$

Where k is the cooling rate of the Simulated Annealing, T is set at the start to the value T_0 then is updated according to the following formula:

$$T \leftarrow \frac{T}{1+(T * fractioning)}$$

Chapter 4

Computational results

4.1 Simulation parameters

Simulation Parameters (Based on Keskin et al. [1]) in Table 4.1:

Table 4.1: Simulation Parameters

Category	**Parameter**	**Value**
Energy-related	Unit energy cost	0.4
	Driver Wage	1
	Fixed Vehicle Acquisition Cost	1200
	Overtime Factor	11/6
Station Utilization	Utilization Range	$\rho_{low} = 0.3$, $\rho_{high} = 0.7$
Customers Service Time	R	Low = 8, High = 12
	C	Low = 70, High = 110
	RC	Low = 8, High = 12

Solver Parameters in Table 4.2:

These parameters collectively define the simulation environment and the solver's behavior. The energy-related parameters, station utilization, and service times influence the characteristics of the simulated system. In contrast, the solver parameters control the annealing process, ALNS behavior, and other aspects of the optimization algorithm.

4.2 Performance metrics

In-Sample Stability

The problem is solved multiple times with different seed values. The more the solutions and the objective functions are similar, the better. The result will be represented by OF value vs. the number of scenarios with standard deviation boxes. In Fig. 4.1 we can see that the problem provides a stable solution regardless of the number of scenarios considered, even when input conditions differ.

The stability, both in and out of sample, was computed on instance $r105\_21\_100.txt$ with 50 scenarios and the mean of 10 simulations for every number of scenarios.

Table 4.2: Solver Parameters

Category	**Parameter**	**Value**
Annealing Parameters	Initial Temperature (T_0)	300
	Cooling Rate (k)	2
	Fractioning	0.1
ALNS Scores	Global Best	5
	Current Best	3
	Solution Accepted	1
	Solution Rejected	0
ALNS Parameters	Number of Zones	4
	Decay Parameter	0.9
	N_{it}	5000
	N_{it} for Station Removal	5
	N_{it} for Weight Update	5
Shaw Parameters	λ	9
	μ	3
	ν	2
	ϕ	5
Destruction Operators Determinism Factors	Worst Removal	2
	Shaw Removal	2
Greedy Confidence	Confidence Level	0.8
Destroy Parameters	γ_c (Number customers removed)	3
	γ_s (Number stations removed	1
State of Charge Tolerance	Tolerance	0.15

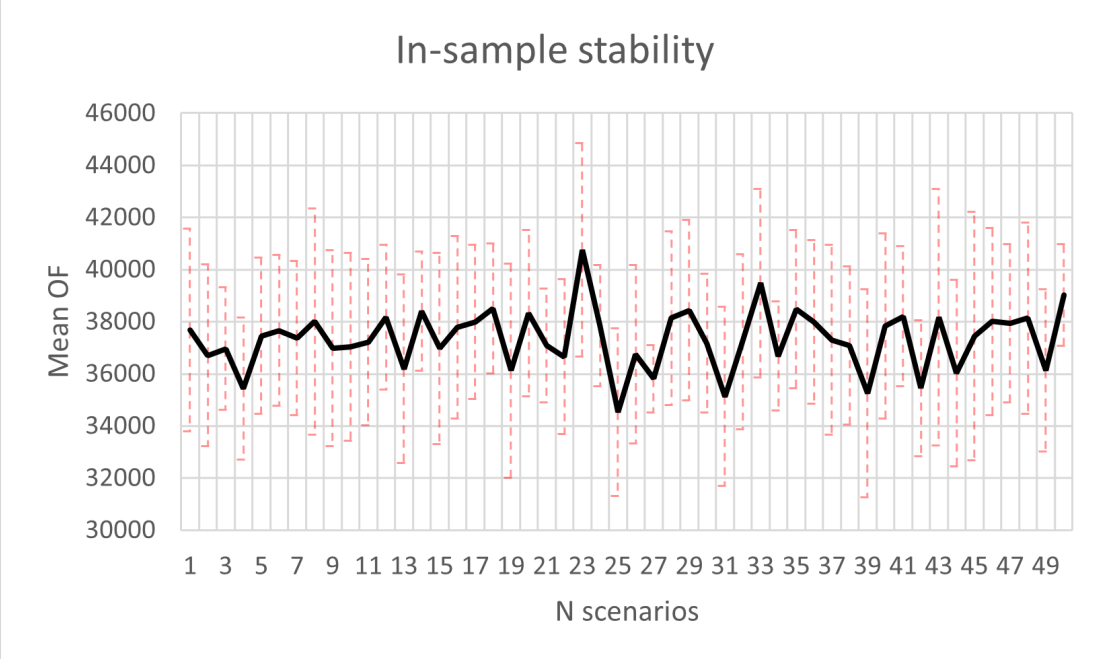

Figure 4.1: In-sample stability

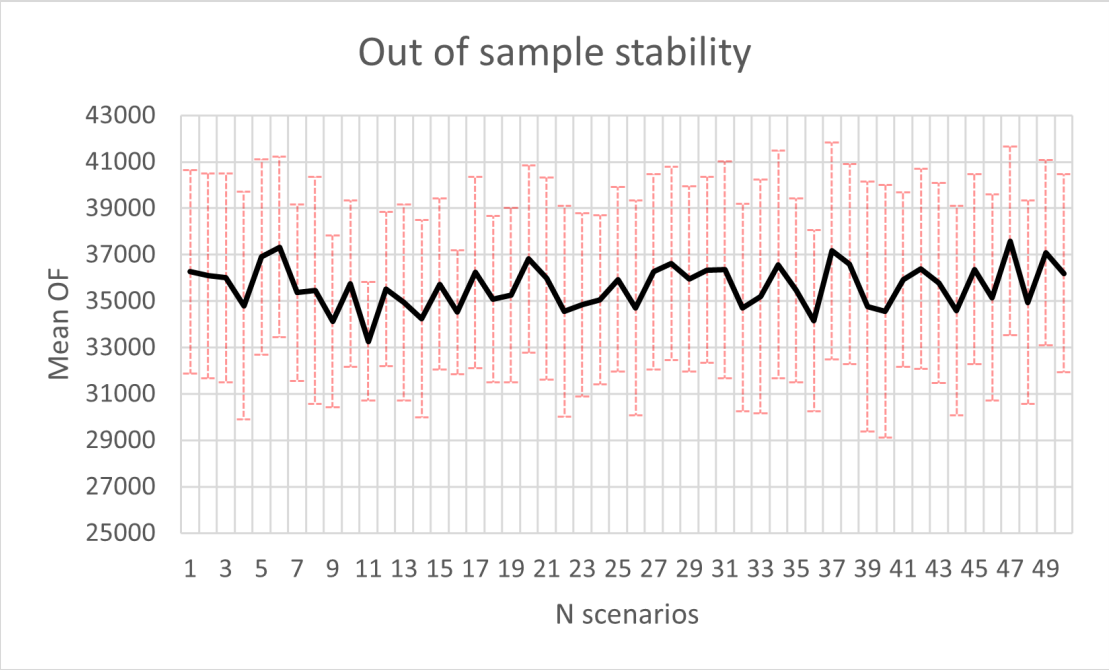

Figure 4.2: Out of sample stability

Out-of-Sample Stability

The problem is solved once, then the instance is generated with new waiting time values and the solution of the first problem is applied to the new realizations.

- solve the problem and take the first stage solution
- generate several scenarios
- for each of them, compute the second stage

The result is similar to the in-sample stability as proven in Fig. 4.2.

4.3 The value of pickup sharing and partial recharge

The primary focus of this thesis was to replicate and build upon the heuristic approach proposed by Keskin et al. (2021)[1] for the Electric Vehicle Routing Problem with Time Windows (EVRPTW). Expanding upon their methodology and introducing two new recourse actions aimed at achieving significant cost reductions in terms of the objective function.

The results of the computational experiments, as summarized in Table 4.3, indicate that, on average, the proposed algorithm does not consistently outperform the performance of the solutions obtained by Keskin et al. However, upon closer examination of Table 4.4, it becomes evident that in the best-case scenarios, the algorithm proposed in this thesis tends to outperform Keskin's approach in most cases.

Figure 4.4 offers a clearer perspective on how the algorithm's performance varies, specifically when analyzing the outcomes across different recourse actions and their application to various

instance types. It reveals that for R and RC class instances, the algorithm not only consistently outperforms the results obtained by Keskin et al., but it also shows a remarkable consistency in its performance. This is evident through the tightly grouped results, indicating that regardless of the recourse action applied, the algorithm maintains a stable and predictable level of performance for these classes.

On the other hand, the behavior of the algorithm when dealing with C-class instances is notably more erratic. The performance fluctuates significantly across different strategies, underscoring the complexity and challenges associated with these particular instances. This variability in outcomes highlights the C class instances as being particularly challenging for the algorithm, both in terms of achieving consistent objective function values and in managing computational times efficiently.

This analysis thus underscores a clear dichotomy in the algorithm's performance: while it exhibits robustness and reliability when applied to R and RC classes, achieving consistently better outcomes than Keskin's approach with low variability, it struggles to maintain this level of performance across the more complex C class instances. The unpredictability associated with the C class instances points to a need for further optimization or a tailored approach to address the inherent complexities of these scenarios, ensuring both efficiency in computation and consistency in achieving high-quality solutions.

This nuanced performance can be further appreciated through the distribution patterns observed in Figure 4.3, which illustrates the algorithm's behavior across different types of instances. The performance distribution for R and RC classes closely resembles a right-skewed Gaussian distribution, characterized by a tail that extends towards higher values but with fewer instances reaching these peaks. This indicates that while there are outliers with higher performance, the bulk of these instances are concentrated towards the lower end of the scale, suggesting a level of consistency in performance with fewer instances of exceptional outcomes.

Conversely, the performance across C class instances appears more variable and less predictable. This variability results in a higher average cost across all tests for the C class, pointing to a scenario where the algorithm's behavior is less consistent and more susceptible to fluctuations. Such randomness in performance suggests that while the algorithm can achieve high marks, it does so with less reliability, particularly in the context of C-class instances where it demonstrates a wider range of outcomes.

In summary, while the algorithm might not consistently outshine the benchmark set by Keskin et al. across all tests, it shows a noteworthy capacity to achieve superior results in optimal conditions. The performance across different instance classes further indicates that the algorithm's efficacy varies, with more predictable outcomes for R and RC classes and a broader performance spectrum for C class instances, highlighting its adaptability and potential for high achievement in specific contexts.

It's important to highlight that introducing additional recourse actions plays a crucial role in improving the overall cost. By offering more diverse options for addressing unforeseen events in the routing process, the algorithm becomes more adept at handling the second step of the problem, leading to improvements in the total cost. It's worth noting that while this improvement is observed across many instances, it's not universal, and variance in performance can be attributed to the inherent stochastic nature of the problem.

In summary, although the proposed algorithm may not consistently outperform Keskin's approach regarding mean performance, it demonstrates superior performance in certain scenarios. The enhanced capability to handle unexpected events through diversified recourse options contributes to improved total cost in many cases, underscoring the effectiveness of the proposed methodology.

In evaluating the computation times as detailed in Table 4.5, it becomes clear that the algorithm in question demonstrates a high level of efficiency, completing tasks in a relatively

brief period. However, the time required for computations is not uniform across all scenarios. Specifically, scenarios with clustered time windows stand out for requiring more processing time—often four to five times as much—as compared to other types. This observation is further supported by the distribution of computational times for C class instances, as illustrated in Figure 4.5, where most computational times cluster between 200 and 400 seconds, indicating a broader spread across the spectrum of possible values. In contrast, R and RC class instances mainly exhibit computational times within the 0-200 second range, with occurrences beyond 200 seconds being notably less frequent. This pattern suggests that the algorithm's performance is more consistent and efficient for R and RC classes, whereas it encounters variability and longer processing times with C class instances.

Despite this discrepancy, it's essential to contextualize these time requirements within real-life applications. The overall computational demand remains insignificant even in scenarios where the algorithm takes longer to converge, such as with clustered time instances. Practically, a simulation lasting around 20 minutes is often sufficient to derive a highly satisfactory solution for the given routing problem.

Therefore, while there may be variations in computational time across different instance types, the algorithm's overall efficiency remains commendable and well-suited for real-world applications, where timely decision-making is crucial.

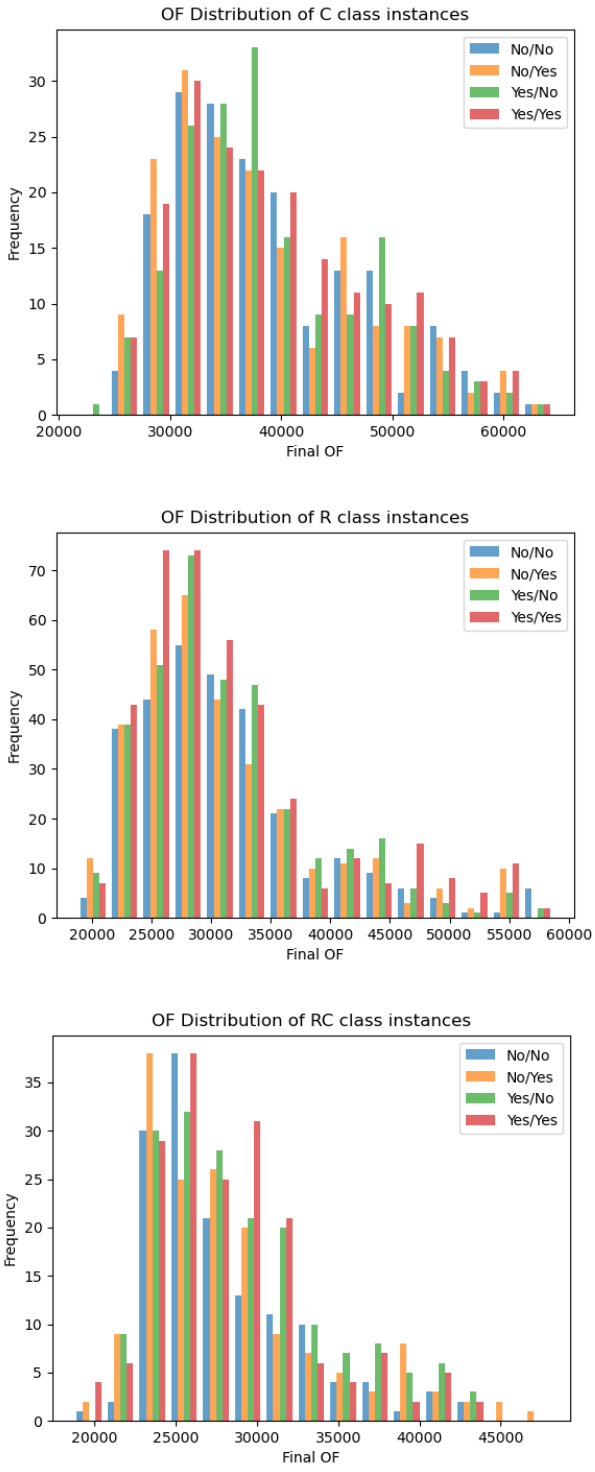

Figure 4.3: Objective Function distribution of values over different kinds of recourse divided for the class of instances

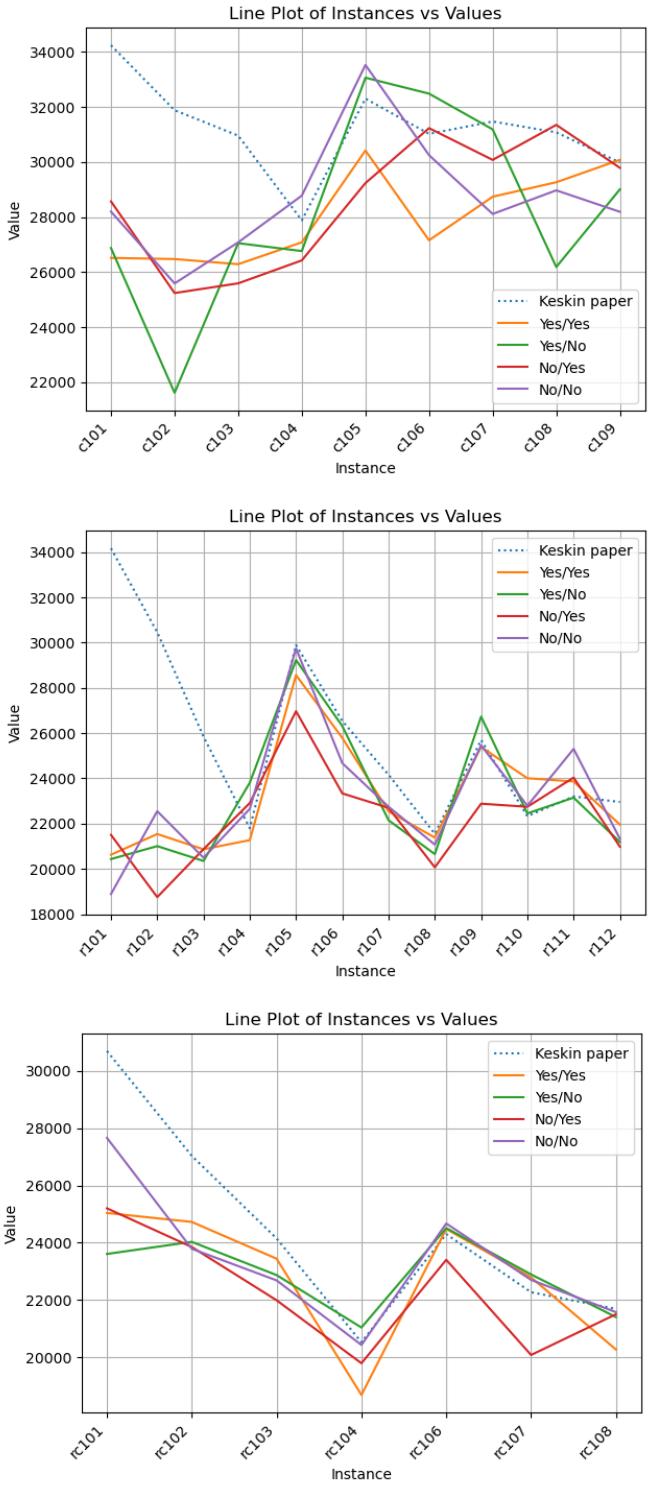

Figure 4.4: Objective Function minimum values vs Keskin results over different instances grouped by class

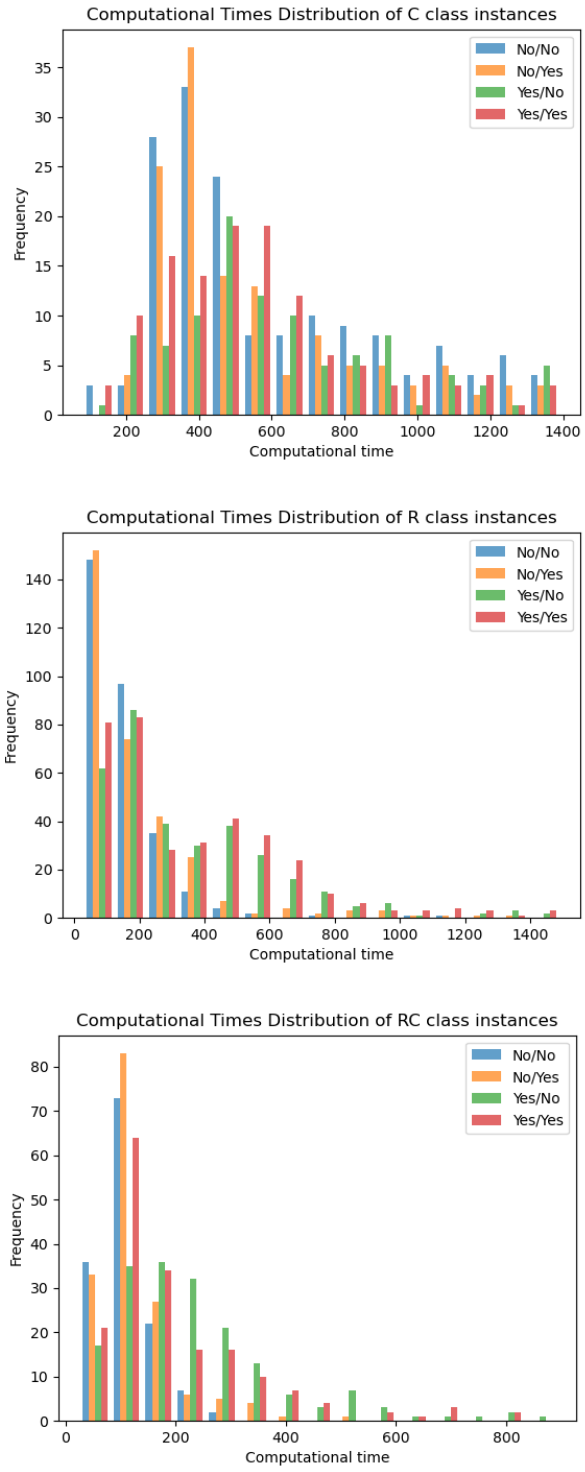

Figure 4.5: Computational times distribution of values over different kinds of recourse divided for the class of instances

Table 4.3: Mean Objective Function computational results obtained from the combination of the usage of the alternative recourse scenarios. The first word refers to pickup exchange, the second word refers to partial recharge (e.g. No/No means no pickup exchange and no partial recharge)(number of customers = 100)

Instance name	Yes/Yes	Yes/No	No/Yes	No/No	Keskin paper
c101	41314	37425	37030	41295	34245
c102	40504	42161	42404	39218	31885
c103	38336	35217	36817	36742	30963
c104	32679	31992	31023	32491	27886
c105	42891	44033	41768	45237	32307
c106	40829	43055	42197	38989	31025
c107	40311	40426	39313	39812	31476
c108	37221	38102	37817	37811	31082
c109	37385	35429	35946	35603	29985
r101	37195	34729	38290	37062	34167
r102	32400	33988	32754	33896	30458
r103	28832	28072	27754	31054	25848
r104	28290	29343	27713	28284	21796
r105	34771	36635	34903	37616	29891
r106	31775	31754	32418	32050	26536
r107	29591	28831	27952	28914	24159
r108	25430	26377	25754	25931	21548
r109	29749	30657	28496	30608	25692
r110	26861	26786	26805	26736	22328
r111	27865	27230	27627	27226	23202
r112	23636	22967	23048	23492	22955
rc101	31682	33457	33253	33546	30696
rc102	32357	32735	33214	32462	27033
rc103	27033	27587	26896	27107	24147
rc104	23319	23853	23900	24821	20519
rc106	27764	28320	27875	28486	24297
rc107	24858	25373	24521	24968	22268
rc108	23523	23599	23162	24361	21685

Table 4.4: Mean Objective Function computational results obtained from the combination of the usage of the alternative recourse scenarios. The first word refers to pickup exchange, the second word refers to partial recharge (e.g. No/No means no pickup exchange and no partial recharge)(number of customers = 100)

Instance name	Yes/Yes	Yes/No	No/Yes	No/No	Keskin paper
c101	26520	26880	28574	28204	34245
c102	26479	21620	25244	25593	31885
c103	26291	27055	25597	27085	30963
c104	27087	26769	26431	28783	27886
c105	30421	33066	29239	33529	32307
c106	27160	32487	31233	30253	31025
c107	28743	31188	30080	28116	31476
c108	29268	26189	31352	28974	31082
c109	30083	29010	29793	28195	29985
r101	20624	20433	21502	18885	34167
r102	21538	21005	18751	22548	30458
r103	20863	20349	20868	20494	25848
r104	21270	23807	22909	22651	21796
r105	28564	29226	26968	29742	29891
r106	25787	26311	23335	24671	26536
r107	22539	22147	22696	22745	24159
r108	21380	20651	20067	21070	21548
r109	25388	26728	22880	25458	25692
r110	24004	22449	22743	22788	22328
r111	23861	23148	24034	25302	23202
r112	21955	21186	20979	21327	22955
rc101	25038	23605	25199	27660	30696
rc102	24723	24027	23848	23786	27033
rc103	23438	22866	21985	22681	24147
rc104	18678	21033	19785	20424	20519
rc106	24480	24510	23400	24670	24297
rc107	22788	22888	20073	22698	22268
rc108	20266	21399	21496	21571	21685

Table 4.5: Mean Objective Function computational times [s] obtained from the combination of the usage of the alternative recourse scenarios. The first word refers to pickup exchange, the second word refers to partial recharge (e.g. No/No means no pickup exchange and no partial recharge)(number of customers = 100)

Instance name	Yes/Yes	Yes/No	No/Yes	No/No
c101	1185	1603	1058	694
c102	1104	1376	696	631
c103	1213	1106	955	607
c104	1502	727	952	521
c105	1557	1340	888	587
c106	1265	1658	970	597
c107	1092	1272	987	765
c108	1260	2274	913	800
c109	2003	2115	1187	953
r101	1206	759	394	256
r102	721	785	422	153
r103	320	291	159	156
r104	241	203	123	117
r105	199	209	137	89
r106	244	312	132	156
r107	230	266	112	121
r108	405	334	172	165
r109	246	228	125	108
r110	261	240	106	131
r111	244	258	167	151
r112	239	230	169	119
rc101	198	245	127	117
rc102	243	294	137	117
rc103	196	222	113	117
rc104	182	244	104	103
rc106	135	164	118	90
rc107	153	197	117	118
rc108	226	214	156	115

Chapter 5

Conclusions and future improvements

The primary goal of this thesis was to replicate and enhance the heuristic approach proposed by Keskin et al. (2021)[1] and to introduce additional recourse steps to improve the solution. This objective was successfully achieved, resulting in an improved solution methodology and providing a framework for potential future enhancements.

One of the most challenging aspects of this endeavor was the development of the solver and the fine-tuning of all aspects related to the Adaptive Large Neighborhood Search (ALNS) algorithm, including the implementation of various destroy and repair methods. It was crucial to ensure that the code remained efficient and could execute within a reasonable time frame.

Looking ahead, there are several avenues for further exploration and enhancement. This includes exploring different approaches to constructing initial solutions, leveraging machine learning techniques to augment the ALNS algorithm's performance, expanding the repertoire of recourse actions, and refining the modeling of charging stations, batteries, and chargers.

The field of route optimization continues to attract increasing attention, driven by the pressing need to reduce carbon emissions in the transportation sector. Advancements in vehicle technology and overall operational efficiency play a crucial role in achieving this objective. As such, ongoing research and innovation in route optimization hold significant promise for contributing to sustainability efforts in the transport industry.

Bibliography

[1] Merve Keskin, Bülent Çatay, and Gilbert Laporte. «A simulation-based heuristic for the electric vehicle routing problem with time windows and stochastic waiting times at recharging stations». In: *Computers & Operations Research* 125 (Jan. 2021), p. 105060. DOI: 10.1016/j.cor.2020.105060. URL: https://doi.org/10.1016/j.cor.2020.105060 (cit. on pp. 10, 11, 13–17, 32, 34, 36, 45).

[2] Roberta Quadrelli and Sierra Peterson. «The energy–climate challenge: Recent trends in CO2 emissions from fuel combustion». In: *Energy Policy* 35.11 (2007), pp. 5938–5952. ISSN: 0301-4215. DOI: https://doi.org/10.1016/j.enpol.2007.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0301421507003126 (cit. on p. 10).

[3] Enzo Barberio Mariano, José Alcides Gobbo, Flávia de Castro Camioto, and Daisy Aparecida do Nascimento Rebelatto. «CO2 emissions and logistics performance: a composite index proposal». In: *Journal of Cleaner Production* 163 (2017). Achieving Low/no Fossil-carbon Economies based upon the Essential Transformations to Support them, pp. 166–178. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2016.05.084. URL: https://www.sciencedirect.com/science/article/pii/S0959652616305480 (cit. on p. 10).

[4] Juliane Müller. «Approximative solutions to the bicriterion Vehicle Routing Problem with Time Windows». en. In: *Eur. J. Oper. Res.* 202.1 (Apr. 2010), pp. 223–231 (cit. on p. 13).

[5] O Arbelaitz, C Rodriguez, and I Zamakola. «Low cost parallel solutions for the VRPTW optimization problem». In: *Proceedings International Conference on Parallel Processing Workshops*. Valencia, Spain: IEEE Comput. Soc, 2002 (cit. on p. 13).

[6] Ibrahim Hassan Osman. «Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem». en. In: *Ann. Oper. Res.* 41.4 (Dec. 1993), pp. 421–451 (cit. on p. 13).

[7] G Clarke and J W Wright. «Scheduling of vehicles from a central depot to a number of delivery points». en. In: *Oper. Res.* 12.4 (Aug. 1964), pp. 568–581 (cit. on p. 13).

[8] Marius M Solomon. «Algorithms for the vehicle routing and scheduling problems with time window constraints». en. In: *Oper. Res.* 35.2 (Apr. 1987), pp. 254–265 (cit. on p. 13).

[9] Jean-Yves Potvin and Jean-Marc Rousseau. «An exchange heuristic for routeing problems with time windows». In: *J. Oper. Res. Soc.* 46.12 (Dec. 1995), pp. 1433–1446 (cit. on p. 13).

[10] Tomislav Erdelić and Tonči Carić. «A survey on the electric vehicle routing problem: Variants and solution approaches». en. In: *J. Adv. Transp.* 2019 (May 2019), pp. 1–48 (cit. on p. 14).

[11] Angel Juan, Carlos Mendez, Javier Faulin, Jesica de Armas, and Scott Grasman. «Electric Vehicles in Logistics and Transportation: A Survey on Emerging Environmental, Strategic, and Operational Challenges». In: *Energies* 9.2 (Jan. 2016), p. 86. ISSN: 1996-1073. DOI: 10.3390/en9020086. URL: http://dx.doi.org/10.3390/en9020086 (cit. on p. 14).

[12] Dimitris Margaritis, Afroditi Anagnostopoulou, Alkiviadis Tromaras, and Maria Boile. «Electric commercial vehicles: Practical perspectives and future research directions». In: *Research in Transportation Business & Management* 18 (Mar. 2016), pp. 4–10. ISSN: 2210-5395. DOI: 10.1016/j.rtbm.2016.01.005. URL: http://dx.doi.org/10.1016/j.rtbm.2016.01.005 (cit. on p. 14).

[13] Samuel Pelletier, Ola Jabali, and Gilbert Laporte. «50th Anniversary Invited Article—Goods Distribution with Electric Vehicles: Review and Research Perspectives». In: *Transportation Science* 50.1 (Feb. 2016), pp. 3–22. ISSN: 1526-5447. DOI: 10.1287/trsc.2015.0646. URL: http://dx.doi.org/10.1287/trsc.2015.0646 (cit. on p. 14).

[14] Timothy M. Sweda, Irina S. Dolinskaya, and Diego Klabjan. «Adaptive Routing and Recharging Policies for Electric Vehicles». In: *Transportation Science* 51.4 (Nov. 2017), pp. 1326–1348. ISSN: 1526-5447. DOI: 10.1287/trsc.2016.0724. URL: http://dx.doi.org/10.1287/trsc.2016.0724 (cit. on p. 14).

[15] Nicholas D. Kullman, Justin C. Goodson, and Jorge E. Mendoza. «Electric Vehicle Routing with Public Charging Stations». In: *Transportation Science* 55.3 (May 2021), pp. 637–659. ISSN: 1526-5447. DOI: 10.1287/trsc.2020.1018. URL: http://dx.doi.org/10.1287/trsc.2020.1018 (cit. on p. 14).

[16] Hongtao Lei, Gilbert Laporte, and Bo Guo. «A generalized variable neighborhood search heuristic for the capacitated vehicle routing problem with stochastic service times». en. In: *TOP* 20.1 (Apr. 2012), pp. 99–118 (cit. on p. 15).

[17] F Errico, G Desaulniers, M Gendreau, W Rei, and L-M Rousseau. «A priori optimization with recourse for the vehicle routing problem with hard time windows and stochastic service times». In: *Eur. J. Oper. Res.* 249.1 (Feb. 2016), pp. 55–66 (cit. on p. 15).

[18] Michel Gendreau, Ola Jabali, and Walter Rei. «Chapter 8: Stochastic vehicle routing problems». In: *Vehicle Routing*. Philadelphia, PA: Society for Industrial and Applied Mathematics, Nov. 2014, pp. 213–239 (cit. on p. 15).

[19] *Solomon VRP instances*. http://www.vrp-rep.org/datasets/item/2014-0013.html. Accessed: 2022-09-28 (cit. on p. 16).

[20] Guy Desaulniers, Fausto Errico, Stefan Irnich, and Michael Schneider. «Exact Algorithms for Electric Vehicle-Routing Problems with Time Windows». In: *Operations Research* 64.6 (Dec. 2016), pp. 1388–1405. DOI: 10.1287/opre.2016.1535. URL: https://doi.org/10.1287/opre.2016.1535 (cit. on p. 16).