Erasmus University Rotterdam

Capacitated charging stations in the electric heterogeneous fleet pickup and delivery problem with time windows

Master Thesis: Operations Research and Quantitative Logistics

Author:
Dani Leidelmeijer (428872)

Supervisor:
dr. T.A.B. Dollevoet

Second assessor:
M.A. van Zon MSc

January 14, 2021

Abstract

The operations research area has shown an increasing interest in (variants of) Green Vehicle Routing Problems. One variant is the Pickup and Delivery Problem with Time Windows and Electric Vehicles. However, existing research on this problem has not considered the possibility of a fleet consisting of different types of vehicles. Also, existing research on this problem has not considered the possibility that conflicts might occur at the stations at which the vehicles need to recharge. Therefore, we introduce the Pickup and Delivery Problem with Time Windows, a Heterogeneous fleet of Electric Vehicles and Capacitated charging stations, in which both of these aspects are taken into account. We provide a mathematical formulation of the problem and also a heuristic to solve realistically sized instances. We compare the heuristic, which is an Adaptive Large Neighbourhood Search, to an exact method and show that optimal results are difficult to obtain. Next, we compare a partial- to a full recharge policy and conclude that the former is superior. Lastly, we quantify the differences in obtained solutions as the number of stations and the rate at which the vehicles recharge decrease.

Contents

1 Introduction **3**

2 Literature Review **5**
2.1 The G-VRP and extensions 5
2.2 The G-PDP and extensions 6

3 The pickup and delivery problem with time windows, heterogeneous electric vehicles and capacitated charging stations **8**
3.1 Problem description 8
3.2 Two classes of exact formulations 9
3.3 Path-based formulation 10
3.3.1 Defining the problem on a graph 10
3.3.2 Generating the set of arcs 11
3.3.3 Dominance rules and preprocessing steps 12
3.3.4 Mixed integer program 13
3.3.5 Conflict-free constraints 16

4 Adaptive Large Neighborhood Search for the PDPTW-HEVC **18**
4.1 ALNS heuristic 18
4.1.1 ALNS procedure 18
4.1.2 Minimal recharge assumption 21
4.2 Construction heuristic 22
4.3 Destroy methods 23
4.3.1 Request removal 23
4.3.2 Route removal 25
4.3.3 Hybrid CS and request removal 25
4.4 Repair methods 26
4.5 Detect and resolve conflicts 26

5 Numerical studies **30**
5.1 Test instances 30
5.2 Implementation issues influencing the comparison 31
5.3 Parameter tuning 32
5.4 Results 33
5.4.1 ALNS compared to exact formulation 33
5.4.2 Partial- compared to full recharge policy 35
5.4.3 Conflict analysis 37

6 Conclusion **41**

A Floyd-Warshall algorithm and backtracking procedure **46**

B Construction heuristic for ALNS **48**

C Parameter tuning process **49**

D Notes on Java code **50**

1 Introduction

In the past decade, the logistics sector has started showing an increased interest in the green aspect of logistics, which takes into account environmental and social consequences that are involved (Schneider, Stenger, and Goeke (2014)). Companies might have several reasons for caring about these consequences, among which are improving the company's image and obeying to governmental regulations. Transportation is an important area of green logistics, because it has several negative consequences such as the emission of greenhouse gasses (GHGs) and production of noise. Reducing these consequences has become more and more popular. As an example, the European Union continues to come up with climate strategies and targets to reduce the GHG emission in Europe.

The trend in green logistics can also be observed in the operations research area. A commonly studied problem related to green logistics is the Green Vehicle Routing Problem (G-VRP). This problem extends the classical Vehicle Routing Problem (VRP), in which a set of customers needs to be assigned to a set of vehicles in an optimal way. The G-VRP is concerned with the additional challenges that result from using a fleet of Alternative Fuel Vehicles (AFVs) instead of conventional (combustion engine) vehicles (Erdoğan and Miller-Hooks (2012)). Electric vehicles (EVs), which belong to the class AFV, have been the topic of many research papers. The variant of the VRP that focuses on these vehicles in particular is called the Electric VRP (E-VRP). In general, EVs are known for their sustainable fuel source, absence of local GHG emission and reduced noise production, which makes them a promising alternative in green logistics. On the other hand, they are known to have a high purchasing cost and decreased driving range. The latter is especially challenging, given the state of the current recharge infrastructure as well as the relatively long recharge time for an EV. Nevertheless, we observe continuous technological progress trying to alleviate these disadvantages and EVs are already being used in present day transportation of last-mile deliveries.

This research is concerned with the challenges that follow from using EVs in a close variant of the VRP, namely the Pickup and Delivery Problem (PDP). One of the first to study the generic PDP were Dumas, Desrosiers, and Soumis (1991), as they introduced the Pickup and Delivery Problem with Time Windows (PDPTW). In the problem, a set of transportation requests is given, each requiring a pickup and corresponding delivery at specified locations. The time windows indicate the time intervals wherein these pickups and deliveries must occur. The problem has important practical implications for both goods- and passenger transportation (Goeke (2019)). The PDP generalizes the VRP in the sense that the problems are equal if either all pickups or all deliveries are located at the central depot, ceteris paribus. As the VRP is well-known to be NP-hard, its generalization is too and therefore existing research often relies on heuristics to solve realistically sized instances of the PDP and the PDPTW.

The use of EVs in the PDPTW is generally described by the PDPTW-EV. In this problem, a fleet of EVs having a limited battery capacity is available for the transportation requests. Given this limited battery capacity, these vehicles may need to recharge at charging stations (CSs). Existing research on the PDPTW-EV has not considered the possibility of a heterogeneous fleet of EVs. In practice the homogeneity assumption might not hold true as the vehicles might differ in terms of freight capacity, battery capacity and the like. Also, existing research assumes that the CSs are always available. However, as CSs are relatively scarce in the current recharge infrastructure, we can imagine that conflicts might occur. These conflicts affect the feasibility of route plans, as they could cause delays such that subsequent customers cannot be visited in time. Therefore, we will provide a solution method for the PDPTW-HEVC, the Pickup and Delivery Problem with Time Windows, a Heterogeneous fleet of Electric Vehicles and Capacitated charging stations. Given a heterogeneous fleet of EVs and a set of transportation requests, the problem consists of finding an optimal set of routes such that the time windows are respected and the capacity on the CSs is never exceeded.

We will provide a mathematical formulation for the problem with which one can try to solve small-sized, manageable instances to optimality, using a commercial solver like CPLEX. As the PDPTW-HEVC extends the PDPTW, an Adaptive Large Neighborhood Search (ALNS) metaheuristic will be provided to solve larger, more realistically sized instances efficiently. To demonstrate the performance of our proposed solution methods, we will generate a set of small instances and a set of large instances. The solutions to the small instances, resulting from the exact formulation, will serve as a benchmark to assess the performance of our metaheuristic. Using the large instances, we will quantify the differences between a partial recharge- and a full recharge policy. Under the former policy, an arbitrary amount can be recharged at a station, while under the latter the amount recharged must always equal the difference between maximum capacity and the remaining battery level. Also, we will study the effects on the obtained solutions and the number of conflicts, as we decrease the number of CSs and the speed with which EVs recharge.

The remainder of this paper is organized as follows: In Section 2, we discuss existing research on the G-VRP and the G-PDP. In Section 3, we provide a detailed description together with a path-based formulation of the problem PDPTW-HEVC, while Section 4 is devoted to a description of the ALNS metaheuristic. Our experimental results are presented in Section 5. Finally, concluding remarks are given in Section 6.

2 Literature Review

Green routing problems have received considerable interest from the operations research community. Even though the VRP and the PDP are different problems, they share a lot of similar challenges regarding the successful introduction of AFVs and EVs. Examples include dealing with a heterogeneous fleet, choosing an efficient recharge policy and avoiding conflicts at the CSs, as we will discuss in this section. First, we will elaborate upon the G-VRP and its extensions (Section 2.1) after which we will discuss research on the green PDP (G-PDP) and its extensions (Section 2.2).

2.1 The G-VRP and extensions

Among the first to introduce a limited driving range and the need to refuel for the VRP were Conrad and Figliozzi (2011). The authors studied the Recharging VRP (RVRP) in which a homogeneous fleet of vehicles has a limited driving range and is allowed to refuel at customer locations. There are no separate locations for refuelling stations and the refuelling time is assumed to be constant, independent of the current level of fuel upon arrival at the customer. Refuelling is executed up until a certain percentage of the maximum capacity. Furthermore, it is assumed that the customer service operations and the refuelling operations occur simultaneously. While time windows are considered, capacity constraints are not considered. The assumption of refuelling at customer locations only was relaxed by Erdoğan and Miller-Hooks (2012), who formally introduced the G-VRP. In this problem, the available fleet of AFVs also had a limited driving range such that there was a need to refuel at specified locations. The authors propose two heuristics to solve realistically sized instances of the problem. The first is a Modified Clarke and Wright Savings (MCWS) algorithm which creates feasible routes by inserting fuelling stations, then merges feasible routes and removes redundant stations. The second heuristic is a Density-Based Clustering Algorithm (DBCA) which clusters customers based on predefined densities. As a recharge policy, the homogeneous AFVs are assumed to recharge fully at a station and the refuelling time is assumed to be constant. Also, the authors do not consider time windows or capacities on the vehicles.

One of the first authors to focus on EVs in particular were Schneider, Stenger, and Goeke (2014), as they introduced the Electric Vehicle Routing Problem with Time Windows and recharging stations (E-VRPTW). Contrary to earlier works, in this problem the recharge time at designated stations depends on the remaining battery level upon arrival. Also, both time windows and vehicle capacities are respected. However, it is still assumed that the fleet of vehicles is homogeneous and always recharges fully at a station. The authors propose a hybrid between Variable Neighborhood Search (VNS) and Tabu Search (TS) as a solution method for the problem. In a similar work by Felipe et al. (2014) who also focus on EVs in the VRP, two important extensions are provided. First

of all, the authors introduce the possibility to perform a partial recharge at a station, motivated by the time- and cost savings that could arise from this. As an example, a partial recharge could help a vehicle reach a subsequent time window in time. Also, the amount recharged too much in a full recharge policy may be recharged at the depot at a cheaper cost. As a second extension, different recharge technologies are introduced, which imply varying recharge times and costs. Deterministic local search heuristics, strengthened with a Simulated Annealing (SA) algorithm, are proposed as a solution method. The authors assume that the fleet of EVs is homogeneous and do not consider time windows. Keskin and Çatay (2016) quantify and confirm the benefits of using a partial recharge policy.

Another important extension to the E-VRP was provided by Ding, Batta, and Kwon (2015), who relaxed the assumption of an unlimited availability of the CSs. As queues at the stations impact the feasibility of the routing plan in terms of time windows at subsequent customers or total route duration, considering the unavailability of CSs is an important challenge. The authors assume a fixed capacity of one EV at every station which may never be exceeded. To make sure the capacity is never exceeded, the authors introduce a set of conflict-free constraints. These constraints ensure that there is never any overlap between the time intervals at which the EVs recharge at the CSs. As a solution method, a variant on the hybrid heuristic proposed by Schneider, Stenger, and Goeke (2014) is given. The authors assume a homogeneous fleet of EVs. In a later work by Froger et al. (2017), the concept of capacitated CSs was also studied, but with a variable capacity on every station. The authors propose a two-stage solution method in which the capacity constraints are first relaxed and then a variant on Benders decomposition is used to introduce capacity on the stations. Also, the concept of variable capacities on the stations was proposed by Bruglieri, Mancini, and Pisacane (2019).

Among the first to introduce a heterogeneous fleet of EVs were Hiermann et al. (2016). The authors proposed the Electric Fleet Size and Mix Vehicle Routing Problem with Time Windows and Recharging Stations (E-FSMFTW), where the available fleet of EVs differs in terms of freight capacity, battery capacity and acquisition cost. The authors formulate the problem as a set partitioning problem and solve it using branch-and-price. Also, they propose a hybrid metaheuristic combining ALNS with local search and a labelling procedure. Still, the authors assume a full recharge of the batteries only. Importantly, the authors show the benefit of using a fleet of EVs consisting of different types.

2.2 The G-PDP and extensions

While the G-VRP and its extensions have gotten an abundance of scientific attention, the G-PDP has been less studied. The first authors to study a variant of the G-PDP were Grandinetti et al. (2016), as they introduced EVs into the PDPTW. The authors provide a three-objective mathematical formulation for the problem and propose a weighted sum

method (WSM) to solve small instances of the problem. However, no solution method is provided for larger, more realistic instances. Also, the fleet of EVs is assumed to be homogeneous and the authors assume soft time windows. Lastly, the EVs are assumed to recharge fully at a CS.

Recently, Goeke (2019) studied the PDPTW-EV. In this problem, the author allows for a partial recharge policy and compares it to a full recharge policy, concluding that the former policy is superior. Furthermore, he provides a partial recharge strategy in case of soft time windows. In his research, the fleet of EVs is assumed to be homogeneous and a limited capacity on the stations is not considered. The author provides a Granular Tabu Search (GTS) as a solution method for realistically sized instances. GTS is a variant on the well-known Tabu Search and is based on the use of neighborhoods that are reduced drastically in size (Toth and Vigo (2003)). These restricted neighborhoods are obtained from normal neighborhoods by removing moves containing only elements that are unlikely to be part of high quality solutions.

3 The pickup and delivery problem with time windows, heterogeneous electric vehicles and capacitated charging stations

In this section, we give a detailed description of the PDPTW-HEVC (Section 3.1). Also, we discuss two main classes of exact formulations for green routing problems (Section 3.2). Finally, we present a mathematical formulation of the problem (Section 3.3).

3.1 Problem description

In the PDPTW-HEVC, a set of n transportation requests is given, each requiring a pickup $i \in P$ and corresponding delivery $n+i \in D$ of a certain amount of goods q at specified customer locations. That is, we order the pickups and corresponding deliveries such that $P = \{1, .., n\}$ and $D = \{n+1, .., 2n\}$. Every pickup has a positive demand q_i and the corresponding delivery has a negative demand q_{n+i}, satisfying $q_i = -\, q_{n+i}$. Additionally, the start of service time of every customer location $i \in V = P \cup D$ must occur within a specified time interval $[e_i,\, l_i]$ and every $i \in V$ has a service time s_i.

In order to satisfy these requests, a heterogeneous fleet of EVs is available at a central depot. Each request must be performed by the same EV, i.e. there are no transfers between the EVs. Every EV $k \in K$ has a limited freight capacity C^k, which may never be exceeded, and a limited battery capacity Q^k. Also, it has an average speed v^k. Finally, every EV has an acquisition cost f^k. A set F of vertices representing the CSs is given. At each of these CS, an EV is allowed to stop to recharge an amount of energy up to Q^k.

Every customer location $i \in V$ and every CS $f \in F$ are assigned a position in a two-dimensional space where gradients are not considered. The energy consumption between any two points in space is assumed to be linearly related to the distance between those points. Therefore, every EV is assigned a battery consumption rate ρ^k, which represents the reduction of the battery level when traveling one unit of distance. For a more realistic energy consumption model in the E-VRP, taking into account speed, gradients and cargo load, we refer the reader to Goeke and Schneider (2015). We assume that the triangle inequality holds true: in terms of distance, time and energy consumption, it is always more beneficial to travel from point A to point B directly than it is to travel through any point C, which is not on the line segment between A and B.

At every CS, the recharge time is assumed to be linearly related to the amount recharged. Consequently, every EV is given a parameter g^k, the inverse recharge rate, which represents the units of time required to recharge one unit of energy. It should be noted that in reality, the relationship between the time and amount recharged can better be approximated with a concave function from a battery level of about $0.8Q^k$ onward. Therefore, a linear approximation is somewhat optimistic in terms of the time needed to

recharge. For a study on using a piecewise linear approximation in the G-VRP, we refer the reader to Montoya et al. (2017) and Froger et al. (2017).

Additionally in our problem, every CS is assigned a capacity C_f which represents the number of EVs that can simultaneously recharge at the CS. It may not happen that an EV starts its recharge operation at a CS which is fully occupied. Therefore, an EV may arrive at a station which is currently at maximum capacity, but then it has to delay its start of recharge operation until an EV that is currently at the station has left. This is similar to the start of service times at customer vertices: an EV may arrive at a customer even before the earliest time point in the customer's time window, but then it has to wait until this earliest point to start its service.

The objective for the PDPTW-HEVC is to construct a set of routes that begin and end at the central depot and perform one or multiple transportion requests in between, while the sum of total distance travelled and total acquisition cost is minimized. Every request must be covered by exactly one route and the routes must adhere to the beforementioned restrictions related to the battery level, time windows and CS capacities.

3.2 Two classes of exact formulations

Most of the research on green routing problems, whether it be the G-VRP or the G-PDP, has come up with exact formulations in terms of mixed-integer linear programs. In general, the exact formulations for green routing problems can be divided into two classes, both of which we will discuss below.

On the one hand, the problems are modelled using an Arc Flow Formulation (AFF). Erdoğan and Miller-Hooks (2012) introduced such a formulation for the G-VRP and Grandinetti et al. (2016) for the G-PDP. Their formulations are similar to the AFF for the general VRP, which can be modelled on a complete graph with a set of nodes representing the customers and a set of arcs between them. Every arc is assigned a binary decision variable that indicates whether that arc is travelled in the solution. However, the formulations for the G-VRP and the G-PDP are more complicated due to the fact that the CSs are also modelled as nodes in the complete graph. Contrary to customer nodes, these CS nodes may be visited multiple times, or not at all in a solution. As a result, the general structure of the AFF has disappeared and therefore Erdoğan and Miller-Hooks (2012) have introduced dummy nodes (copies) to overcome this issue. In their formulation, every CS is given a limited number of dummies, such that multiple visits to the same CS are possible and the flow conservation constraints are respected. As the authors already noted, choosing an appropriate number of dummies is an important challenge: choosing too few dummies, some feasible and potentially optimal solutions are not considered while choosing too many dummies can significantly increase the complexity of the complete graph and therefore lead to higher runtimes. The number of dummies

is generally chosen as an upper bound on the number of times each CS can be visited. Many research papers have implemented the AFF in a similar way as done by Erdoğan and Miller-Hooks (2012), among which are the works by Schneider, Stenger, and Goeke (2014), Ding, Batta, and Kwon (2015) and Hiermann et al. (2016).

On the other hand, the green routing problems are modelled using a Path Based Formulation (PBF). The first to introduce this formulation were Andelmin and Bartolini (2017) for the G-VRP. They defined a *refuel path* as a simple path in a complete graph, connecting any pair of vertices for customers or the depot, and visiting one or multiple refuelling stations along the way. Introducing the concept of a refuel- or recharge path, the problem may be redefined on a directed multigraph. This multigraph contains an extended set of arcs, such that for every pair of vertices, and for every path between them, an arc is created. The authors introduce dominance rules to leave out recharge paths that can never be part of an optimal solution, thereby limiting the number of arcs in the multigraph. The concept of a recharge path was also used in other studies, among which Froger et al. (2017) and Goeke (2019). A different definition of a recharge path was used by Bruglieri, Mancini, and Pisacane (2019), see Section 3.3. Generally, in existing research every path is assigned a binary decision variable indicating whether that path is travelled. As distinct paths may contain the same CS, this formulation avoids the use of dummies for these CSs while allowing multiple visits to them.

3.3 Path-based formulation

First, we will define the problem on a graph in Section 3.3.1, after which we will explain how we generate an appropriate set of arcs in Section 3.3.2. Also, we will introduce dominance rules and some preprocessing steps in Section 3.3.3. Finally, in Section 3.3.4, we provide a mixed-integer program for the PDPTW-HEVC using a PBF.

3.3.1 Defining the problem on a graph

The customers and copies of the depot $\{0\}$ and $\{2n+1\}$ are represented as vertices in a graph. Whenever a set includes one or both vertices representing the depot instances, we indicate this with a subscript, e.g. $P_{2n+1} = P \cup \{2n+1\}$. To avoid the use of dummies for the CSs, we propose a PBF in which the definition of a recharge path (RP) is similar to that of Goeke (2019). In particular, we define a RP as a simple path between any pair of customer or depot vertices i and j, which visits one or multiple CSs along the way. We define the problem on a directed multigraph $\mathcal{G} = (V_{0,2n+1}, A)$ which consists of a set of vertices $V_{0,2n+1} = V \cup \{0\} \cup \{2n+1\}$ and a set of arcs A. For any pair of vertices and any path between them, we create an arc, i.e. $A = \{(i,j)^h : i \in V_0,\, j \in V_{2n+1},\, h \in H(i,j),\, i \neq j\}$. The set $H(i,j)$ contains all the RPs between customer or depot vertices i and j and also contains a direct path between them.

Definition 1: A RP is a simple path connecting any pair of customer or depot vertices $i \in V_0$ and $j \in V_{2n+1}$, $i \neq j$, while visiting one or multiple CSs f, g, l, ... $\in F$ along the way.

Each arc $(i,j)^h \in A$ has a corresponding distance d_{ij}^h. Also, depending on the EV that travels the path belonging to the arc, every arc has a travel time t_{ij}^{hk} which is obtained by the division d_{ij}^h/v^k. To obtain the energy γ_{ij}^{hk} required to travel a path for an EV, we multiply d_{ij}^h with ρ^k.

3.3.2 Generating the set of arcs

To generate the set A, we use a procedure similar to that of Goeke (2019). We first construct an auxiliary graph on the CSs, $\mathcal{G}^F = (F, A^F)$. In this graph, the vertices represent the CSs while the arcs represent the direct connections between any pair of CSs $(f,\ g) \in F^2$, $f \neq g$. Now, we eliminate all arcs that are not energy feasible for at least one available EV in the set K. That is, an arc $(f,\ g)$ only remains in $\mathcal{G}^F$ if $\exists\ k \in K$ such that $\gamma_{fg}^k \leq Q^k$, where γ_{fg}^k is the energy needed for EV k to travel from CS f to CS g. We assume that a path between a pair of CSs having the smallest distance also has the smallest travel time and energy consumption. Therefore, we search for those paths with minimum distance using the Floyd-Warshall algorithm, which determines the shortest paths in $\mathcal{G}^F$ for every pair of CSs. For a description of the algorithm of Floyd-Warshall and a backtracking procedure, we refer the reader to Appendix A.

To construct the set $H(i,j)$ for any pair of customer vertices i and j, we add the direct path from i to j if it is feasible for at least one available EV in the set K. Next, we add a set of RPs that begin at i, then visit any single CS $f \in F$ and then end in j. Finally, we add a set of RPs that connect i and j through any shortest path in $\mathcal{G}^F$, that is a shortest path between any pair of CSs $(f,\ g) \in F^2$, $f \neq g$. Note that if the auxiliary graph $\mathcal{G}^F$ is complete, i.e. every direct arc is energy feasible for at least one available EV, then the shortest path between any pair of CSs is just the distance of the direct arc between them due to the triangle inequality. While $H(i,j)$ contains both the direct path and the RPs between i and j, we denote with $\widehat{H}(i,j)$ the set of only the RPs. Furthermore, the set $\overline{H}(i,j)$ contains only the direct path.

In our formulation, we implicitly assume that every CS f has a capacity $C_f = 1$. For the introduction of a variable capacity on the stations in the G-VRP, we refer the reader to Bruglieri, Mancini, and Pisacane (2019). The authors introduce a different definition for a RP in the sense that it connects the depot or an Alternative Fuel Station (AFS) with the depot or another AFS.

The total size of all sets $H(i,j)$, and therefore A, is $\mathcal{O}(|V|^2|F|^2)$. However, given some i and j, many RPs in $H(i,j)$ can never be part of an optimal solution. Therefore, existing

research has come up with dominance rules to eliminate those RPs, limiting the size of the multigraph.

3.3.3 Dominance rules and preprocessing steps

We will introduce a definition for dominance similar to earlier works by Andelmin and Bartolini (2017) and Goeke (2019).

Definition 2: Let h and m be two distinct RPs in $H(i,j)$. If f_h and f_m are the first CSs on the RPs, and g_h and g_m are the last CSs on the RPs, then h dominates m if $d_{ij}^h \leq d_{ij}^m \wedge t_{ij}^{hk} \leq t_{ij}^{mk} \wedge \gamma_{ij}^{hk} \leq \gamma_{ij}^{mk} \wedge \gamma_{if_h}^{hk} \leq \gamma_{if_m}^{mk} \wedge \gamma_{g_hj}^{hk} \leq \gamma_{g_mj}^{mk} \; \forall k \in K$.

This definition implies that RP h dominates RP m if they both connect the same pair of customer or depot vertices i and j and it is always more beneficial to travel h than it is to travel m. Therefore, we can stop considering the possibility to travel m altogether. First, the distance of h must be smaller than or equal to the distance of m. Second, the travel time for any EV on h must be smaller than or equal to the travel time of m. This second condition is equivalent to the first condition for a given EV, since the travel time in our model is obtained from dividing the distance by the average speed. Third, the energy consumption of h must be smaller than or equal to that of m. Again, there exists an equivalence relation between this condition and the previous two conditions for a given EV. The fourth and fifth conditions respectively state that the energy consumptions to the first and from the last CS on h must be smaller than or equal to these energy consumptions of m.

Without loss of generality, we assume that all dominated RPs are removed from the set $H(i,j)$. We do not consider the dominance of a direct path because it always dominates a RP that makes a detour to a CS. Elimination would then leave us with only the direct paths. These detours, however, are valuable to recharge our battery. Consequently, we only look at dominance relations within the set of RPs $\widehat{H}(i,j)$.

Next to dominance, we also perform some preprocessing steps to eliminate direct paths or RPs that can never be part of a feasible solution. First of all, we eliminate any path in $H(i,j)$ that connects the start depot with a delivery customer or connects a pickup customer with the end depot, as these paths would hinder proper pickup- and delivery connections. Also, any path connecting the delivery customer with its corresponding pickup customer (in this erroneous order) is removed. These preprocessing steps might seem redundant because they essentially eliminate paths that would otherwise never be chosen due to appropriate constraints. However, the elimination of these paths might be crucial for keeping a manageable runtime even for small instances, as we will discuss in Section 3.3.5.

3.3.4 Mixed integer program

We introduce a number of decision variables for our formulation. Binary variable x_{ij}^{hk} indicates that RP h between customer or depot vertices i and j is travelled by EV k. Variable u_i^k represents the freight level of EV k on arrival at customer or depot vertex i, while ζ_f^{hk} is the amount of energy recharged by EV k at CS f on RP h. Furthermore, variables y_i^k and β_f^{hk} represent the battery level of EV k on arrival at respectively customer or depot vertex i or CS f on RP h. Also, variables τ_i^k and α_f^{hk} are the start of service of EV k on arrival at respectively customer or depot vertex i and CS f on RP h, while δ_f^{hk} represents the end of service at f. For simplicity, we call the recharge operation at a CS a service too. Finally, binary decision variable z_f^{hmkl} indicates whether $\alpha_f^{ml} \geq \delta_f^{hk}$ and is used in a set of constraints that avoid conflicts at the CSs.

In Table 1, an overview is provided of all sets, parameters and variables that are introduced. For notational convenience, we use the abbreviations $h \in H$, $h \in \overline{H}$ and $h \in \widehat{H}$ to indicate that we choose any path from the indicated set for any combination of i and j. As an example, $h \in \widehat{H}$ is substituted for $h \in \widehat{H}(i,j)$, $i \in V_0$, $j \in V_{2n+1}$, $i \neq j$, and represents any RP for any customer or depot vertex pair. A mixed-integer program (MIP) for the PDPTW-HEVC is the following:

$$\min \quad \sum_{k\in K}\sum_{i\in V_0}\sum_{j\in V_{2n+1}}\sum_{h\in H(i,j)} d_{ij}^h x_{ij}^{hk} \tag{1}$$

$$+\sum_{k\in K}\sum_{j\in P_{2n+1}}\sum_{h\in H(0,j)} f^k x_{0j}^{hk} \tag{2}$$

The objective consists of two parts, (1) and (2), and minimizes the sum of total distance travelled and total acquisition cost.

$$\text{s.t.} \quad \sum_{k\in K}\sum_{j\in V\setminus\{i\}}\sum_{h\in H(i,j)} x_{ij}^{hk} = 1 \qquad \forall i \in P \tag{3}$$

$$\sum_{j\in V\setminus\{i\}}\sum_{h\in H(i,j)} x_{ij}^{hk} - \sum_{j\in V\setminus\{i\}}\sum_{h\in H(i,j)} x_{j,n+i}^{hk} = 0 \qquad \forall k \in K, i \in P \tag{4}$$

$$\sum_{j\in P_{2n+1}}\sum_{h\in H(0,j)} x_{0j}^{hk} = 1 \qquad \forall k \in K \tag{5}$$

$$\sum_{i\in D_0}\sum_{h\in H(i,2n+1)} x_{i,2n+1}^{hk} = 1 \qquad \forall k \in K \tag{6}$$

$$\sum_{i\in V_0\setminus\{j\}}\sum_{h\in H(i,j)} x_{ij}^{hk} - \sum_{i\in V_{2n+1}\setminus\{j\}}\sum_{h\in H(i,j)} x_{ji}^{hk} = 0 \qquad \forall k \in K, j \in V \tag{7}$$

Sets (3) - (7) ensure appropriate in- and outflow. In particular, constraint set (3)

Table 1: Sets, parameters and variables in the PDPTW-HEVC.

P	Vertices for pickup locations
D	Vertices for delivery locations
V	Vertices for customer locations, $V = P \cup D$
F	Vertices for rechargers
F^h	Vertices for rechargers on RP h, $F^h \subseteq F$
B_f^h	Vertices for rechargers coming before f on RP h, $B_f^h \subset F$
K	Set of heterogeneous EVs
$H(i,j)$	All paths between vertices $i \in V_0$ and $j \in V_{2n+1}$
$\overline{H}(i,j)$	Set containing the direct path between vertices $i \in V_0$ and $j \in V_{2n+1}$, $\overline{H} \subset H$
$\widehat{H}(i,j)$	All RPs between vertices $i \in V_0$ and $j \in V_{2n+1}$, $\widehat{H} \subset H$
$\widehat{H}_f(i,j)$	RPs between vertices $i \in V_0$ and $j \in V_{2n+1}$ that contain CS f
d_{ij}^h	Distance between vertices i and j on path h
t_{ij}^{hk}	Travel time between vertices i and j on path h by EV k
γ_{ij}^{hk}	Energy consumption between vertices i and j on path h by EV k
C^k	Freight capacity of EV k, i.e. units available for cargo
Q^k	Battery capacity of EV k in units of energy
f^k	Acquisition cost of EV k
g^k	Inverse recharge rate of EV k
q_i	Demand of vertex i in units
s_i	Service time at vertex i in time units
e_i	Earliest start of service at vertex i in time units
l_i	Latest start of service at vertex i in time units
π_f^h	Binary parameter equal to 1 whenever CS f is positioned on RP h
ϕ^h	Binary parameter equal to 1 whenever f is the first CS on RP h
λ_f^h	Binary parameter equal to 1 whenever f is the last CS on RP h
σ_{fg}^h	Binary parameter equal to 1 whenever g succeeds f on RP h
x_{ij}^{hk}	Binary variable indicating that path h between i and j is travelled by k
u_i^k	The freight level of EV k on arrival at vertex i
ζ_f^{hk}	The amount recharged by EV k at CS f on RP h
y_i^k	The battery level of EV k on arrival at vertex i
β_f^{hk}	The battery level of EV k on arrival at CS f on RP h
τ_i^k	The start of service of EV k at vertex i
α_f^{hk}	The start of service of EV k at CS f on RP h
δ_f^{hk}	The end of service of EV k at CS f on RP h
z_f^{hmkl}	Binary variable indicating that $\alpha_f^{ml} \geq \delta_f^{hk}$ for CS f on RP h

ensures that every pickup location is visited exactly once. (4) guarantees that whenever pickup i is visited, the corresponding delivery $n+i$ is also visited and by the same EV. Sets (5) and (6) make sure an EV always starts and ends at the depot vertices. (7) ensures that at every customer vertex the total inflow equals the total outflow.

$$u_i^k + q_i - C^k \left(1 - \sum_{h \in H(i,j)} x_{ij}^{hk}\right) \leq u_j^k \qquad \forall k \in K, i \in V_0, j \in V_{2n+1}, i \neq j \tag{8}$$

$$u_0^k = u_{2n+1}^k = 0 \qquad \forall k \in K \qquad (9)$$

Whenever any path between two vertices is travelled, (8) updates the freight level of an EV. (9) makes sure that at the depot vertices, the freight level is zero.

$$y_i^k - \gamma_{ij}^{hk} + Q^k \left(1 - x_{ij}^{hk}\right) \geq y_j^k \qquad \forall k \in K, h \in \overline{H} \qquad (10)$$

$$\beta_f^{hk} + \zeta_f^{hk} \leq Q^k \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (11)$$

$$y_i^k - \gamma_{if}^{hk} + Q^k \left(1 - \phi_f^h x_{ij}^{hk}\right) \geq \beta_f^{hk} \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (12)$$

$$\beta_f^{hk} + \zeta_f^{hk} - \gamma_{fg}^{hk} + Q^k \left(1 - \sigma_{fg}^h x_{ij}^{hk}\right) \geq \beta_g^{hk} \qquad \forall k \in K, (f,g) \in F^2, f \neq g, h \in \widehat{H} \qquad (13)$$

$$\beta_f^{hk} + \zeta_f^{hk} - \gamma_{fj}^{hk} + Q^k \left(1 - \lambda_f^h x_{ij}^{hk}\right) \geq y_j^k \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (14)$$

$$y_0^k = Q^k \qquad \forall k \in K \qquad (15)$$

Sets (10) - (15) ensure appropriate battery levels. In particular, set (10) updates the battery level in case the direct path is travelled between customer or depot vertices i and j. (11) ensures that at any CS, the amount recharged is less than a full recharge. Whenever a CS is the first station on a RP, (12) updates the battery level accordingly. If CS g succeeds CS f on some RP, (13) updates the battery level from f to g. If some CS is the last station on the RP, (14) updates the battery level at the last vertex on the path.

$$\tau_i^k + t_{ij}^{hk} + s_i - l_{2n+1} \left(1 - x_{ij}^{hk}\right) \leq \tau_j^k \qquad \forall k \in K, h \in \overline{H} \qquad (16)$$

$$e_i \leq \tau_i^k \leq l_i \qquad \forall k \in K, i \in V_{0,2n+1} \qquad (17)$$

$$\tau_i^k \leq \tau_{n+i}^k \qquad \forall k \in K, i \in P \qquad (18)$$

$$\tau_i^k + t_{if}^{hk} + s_i - l_{2n+1} \left(1 - \phi_f^h x_{ij}^{hk}\right) \leq \alpha_f^{hk} \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (19)$$

$$\delta_f^{hk} + t_{fg}^{hk} - l_{2n+1} \left(1 - \sigma_{fg}^h x_{ij}^{hk}\right) \leq \alpha_g^{hk} \qquad \forall k \in K, (f,g) \in F^2, f \neq g, h \in \widehat{H} \qquad (20)$$

$$\delta_f^{hk} + t_{fj}^{hk} - l_{2n+1} \left(1 - \lambda_f^h x_{ij}^{hk}\right) \leq \tau_j^k \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (21)$$

$$\alpha_f^{hk} + g^k \zeta_f^{hk} - l_{2n+1} \left(1 - \pi_f^h x_{ij}^{hk}\right) \leq \delta_f^{hk} \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (22)$$

Sets (16) - (22) ensure appropriate start of service times. In particular, set (16) updates the start of service in case the direct path is travelled between customer or depot vertices i and j. Constraint sets (17) and (18) ensure that the start of service occurs within the time window and every pickup occurs before the corresponding delivery. Sets (19) - (21) update the start of service along a RP. Set (22) relates the start of service to the end of service at a CS.

$$\delta_f^{hk} - \alpha_f^{ml} \leq l_{2n+1}\left(1 - z_f^{hmkl}\right) \qquad \forall (k,l) \in K^2, k \neq l, f \in F, (h,m) \in \widehat{H}_f^2, h \neq m \qquad (23)$$

$$\delta_f^{ml} - \alpha_f^{hk} \leq l_{2n+1} z_f^{hmkl} \qquad \forall (k,l) \in K^2, k \neq l, f \in F, (h,m) \in \widehat{H}_f^2, h \neq m \qquad (24)$$

Constraint sets (23) and (24) make sure that no conflicts occur at any CS and are explained in more detail in Section 3.3.5.

$$x_{ij}^{hk} \in \{0,1\} \qquad \forall k \in K, h \in H \qquad (25)$$

$$C^k \geq u_i^k \geq 0 \qquad \forall k \in K, i \in V_{0,2n+1} \qquad (26)$$

$$\zeta_f^{hk} \geq 0 \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (27)$$

$$y_i^k \geq 0 \qquad \forall k \in K, i \in V_{0,2n+1} \qquad (28)$$

$$\beta_f^{hk} \geq 0 \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (29)$$

$$\tau_i^k \geq 0 \qquad \forall k \in K, i \in V_{0,2n+1} \qquad (30)$$

$$\alpha_f^{hk} \geq 0 \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (31)$$

$$\delta_f^{hk} \geq 0 \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (32)$$

$$z_f^{hmkl} \in \{0,1\} \qquad \forall (k,l) \in K^2, k \neq l, f \in F, (h,m) \in \widehat{H}_f^2, h \neq m \qquad (33)$$

The domains of the decision variables are defined in sets (25) - (33).

If we want to change the partial recharge policy to a full recharge policy, then we need to add the following constraint set to the MIP:

$$\beta_f^{hk} + \zeta_f^{hk} + Q^k(1 - \pi_f^h x_{ij}^{hk}) \geq Q^k \qquad \forall k \in K, f \in F, h \in \widehat{H} \qquad (34)$$

Together with set (11), this constraint set ensures that whenever a CS is located on a RP and that RP is travelled, the amount recharged at this CS equals $\zeta_f^{hk} = Q^k - \beta_f^{hk}$ for any EV. This is equivalent to a full recharge policy.

3.3.5 Conflict-free constraints

At a CS f, a conflict occurs whenever an EV starts its service while the CS is currently occupied. To prevent this, sets (23) and (24) restrict the start and end of service for any two EVs at f. In particular, whenever $\alpha_f^{ml} - \delta_f^{hk} \leq 0$, it must hold that $\alpha_f^{hk} - \delta_f^{ml} \geq 0$ for an EV k travelling RP h and EV l travelling RP m. That is, whenever l starts its service before k ends its service at f, then it must hold that l also ends its service before k starts its service. This way, no conflict can occur at CS f. As every path is travelled by at most one EV due to sets (3) - (7), a conflict can never occur between two EVs travelling the

same path. Also, by definition, an EV can never have a conflict with itself. Consequently, a conflict can only occur for two distinct EVs travelling two distinct RPs. We model the condition that if $\alpha_f^{ml} - \delta_f^{hk} \leq 0$ then $\alpha_f^{hk} - \delta_f^{ml} \geq 0 \; \forall \; (k,\, l) \in K^2,\, k \neq l,\, f \in F,\, (h,\, m) \in \overline{H}_f^2,\, h \neq m$. Using a truth table, we can show that this is equivalent to the condition $\alpha_f^{ml} - \delta_f^{hk} > 0 \vee \alpha_f^{hk} - \delta_f^{ml} \geq 0$. In standard form, we get that $\delta_f^{hk} - \alpha_f^{ml} \leq 0 \vee \delta_f^{ml} - \alpha_f^{hk} \leq 0$. Now this condition can be linearized by introducing the constraint sets (23) and (24).

The sets are defined for all pairs of distinct vehicles, for all CSs, and for all pairs of distinct RPs that contain this CS. The size of this constraint set can thus grow large even for small instances having only a few EVs and a few customers. As a numerical example, one of the small instances with 6 customer vertices and 3 CS vertices has a total of 104 RPs in the set $\widehat{H}$. The total number of RP pairs is therefore $2\binom{104}{2} = 10{,}712$, provided that (h, m) and (m, h) are distinct pairs. Consequently, the total number of constraints in sets (23) and (24) is 82,680. In contrast, the total number of constraints excluding the conflict-free constraints is 9,795, such that the vast majority of constraints belongs to sets (23) and (24). Depending on the application, we might want to speed up our exact method by executing a number of preprocessing steps such as the ones described before, thereby limiting the number of RPs. Some alternative preprocessing steps are given by Goeke (2019).

4 Adaptive Large Neighborhood Search for the PDPTW-HEVC

4.1 ALNS heuristic

The ALNS framework has been introduced by Ropke and Pisinger (2006) in the PDPTW, as a generalization of the Large Neighborhood Search (LNS) framework. ALNS is a generalization in the sense that it allows for the use of multiple destroy- and repair methods. First, a destroy method destroys a part of the solution such that the resulting solution is no longer complete. Then, a repair method rebuilds a partial solution until it is complete. Therefore, every combination of destroy- and repair method corresponds to a neighborhood of a solution. Because multiple methods can be used dynamically, ALNS allows for the use of multiple neighborhoods within the same search procedure, in contrast to LNS.

In a later work, Pisinger and Ropke (2007) have generalized the ALNS framework to different variants of VRPs. Next to applications in general routing problems, the framework has also been applied successfully in green routing problems, e.g. in the E-VRPTW (Keskin and Çatay (2016)) and in the E-FSMFTW (Hiermann et al. (2016)).

4.1.1 ALNS procedure

We define a solution $\mathcal{X}$ as a set of routes. Each route $r \in \mathcal{X}$ is characterized by an EV, which has to drive the route, and by an ordered list of customers that need to be visited. Also, every solution is characterized by the arrival- and departure times at the locations, which depend on the recharge policy used. This is explained in more detail in Section 4.1.2. The general objective is to minimize the sum of total distance travelled and total acquisition cost. For a given solution, let $f(\mathcal{X})$ represent this total sum. We will use function $f(\mathcal{X})$ to evaluate the quality of our obtained solutions in the sense that a solution $\mathcal{X}'$ is at least as good as a different solution $\mathcal{X}$ if it holds that $f(\mathcal{X}') \leq f(\mathcal{X})$.

We denote by Ω^- the set containing the destroy methods, while we denote by Ω^+ the set of repair methods. We incorporate different types of destroy methods: request removal methods, route removal methods and a hybrid combination that removes CSs together with requests. Consequently, the solution resulting from a destroy method is not only infeasible in the sense that not all requests are serviced, but it might also be energy infeasible due to the removal of one or multiple CSs. In contrast to the destroy methods, all repair methods are of the same type in the sense that they aim at inserting unhandled requests one at a time. In every repair method, a specified heuristic for inserting CSs is incorporated, which differs depending on the repair method at hand. This specified heuristic aims at resolving any energy infeasibilities in the route under consideration.

The ALNS algorithm first constructs an initial feasible solution, of which the objective may be far from optimal. The construction heuristic that is used, which is greedy in

nature, is explained in more detail in Section 4.2. We attempt to improve the initial solution iteratively until a maximum number of iterations is reached. Every destroy- and repair method are given a weight, denoted by ρ_d and ρ_r respectively, which controls how often a method is used in subsequent iterations. In particular, we calculate the probability of choosing destroy method $d \in \Omega^-$ using the so-called *roulette wheel principle*. The probability of choosing method d in iteration $t+1$ is based on the weights of all the destroy methods in iteration t and equals

$$\mathcal{P}_d^{t+1} = \frac{\rho_d^t}{\sum_{i=1}^{|\Omega^-|} \rho_i^t}. \tag{35}$$

Here, $|\Omega^-|$ is the number of destroy methods. The probabilities for the repair methods are determined in a similar manner. By modifying the weights dynamically as the search progresses, the heuristic adapts to the specific instance under consideration.

In every iteration of ALNS, the current solution is destroyed by choosing and applying a destroy method. Then, this destroyed solution is repaired by choosing and applying a repair method. After this, we again have a solution in which all requests are serviced. This solution is then tuned by the method *tuneSolution* such that unnecessary visits to CSs and routes visiting no customers are removed.

Then, we check for potential conflicts in the solution using the method *containsConflicts*. If conflicts occur, then the method *resolveConflicts* either resolves all conflicts on all CSs or it returns that the conflicts cannot be resolved. Both methods are explained in more detail in Section 4.5. In case that the conflicts cannot be resolved, ALNS essentially discards the new solution $\mathcal{X}''$ by restarting the current iteration and not updating the current solution or best found solution. That is, we do not make a move towards this neighbouring solution.

In order to escape local minima and diversify our search for a global minimum, we incorporate the commonly used Simulated Annealing (SA) framework into our ALNS. Given a candidate solution $\mathcal{X}''$, which results from destroying, repairing and tuning $\mathcal{X}$ and making $\mathcal{X}$ conflict-free, we decide to occassionaly make a move towards this candidate solution even if it is worse than our current solution. In particular, we always accept better solutions, and if a solution is worse, we calculate the difference in objective $\Delta(\mathcal{X}, \mathcal{X}'') = f(\mathcal{X}) - f(\mathcal{X}'')$. We now accept $\mathcal{X}''$ as our new solution with probability $\exp(-\Delta(\mathcal{X}, \mathcal{X}'')/T)$, where $T < 0$ is a parameter called the temperature. If $\Delta(\mathcal{X}, \mathcal{X}'')$ is large in absolute value, then $\mathcal{X}''$ is much worse than $\mathcal{X}$ and the probability of accepting the candidate solution is small. During the algorithm, we increase T (we make it less negative) such that the probability of accepting worse solutions decreases as the algorithm progresses. In every iteration, T is multiplied by *cooling rate* c, where $0 < c < 1$. Similar to Ropke and Pisinger (2006), we determine an initial value for T based on the problem

instance at hand, using *start temperature control parameter* w. The initial value for T is set such that if a solution is $w\%$ worse than the current solution, it is accepted in the SA framework with probability 0.5.

We incorporate an adaptive weight adjustment procedure similar to Ropke and Pisinger (2006). In particular, we divide the search into a number of segments or learning periods, where each segment consists of n_{seg} iterations. The idea is now to update the weights of the destroy- and repair methods from one segment to another, based on so-called *scores* from these methods. In every iteration one destroy- and one repair method are applied. We update the scores for both of these methods by the same amount, as the quality of the resulting solution is caused by the specific combination of destroy- and repair method. At the beginning of every segment, we assign every method a score of 0. At the end of every iteration in a segment, the scores of the used methods either stay the same, or are updated using score adjustment parameters following the rules presented in Table 2.

Table 2: Score adjustment rules

Parameter	**Rule**
σ_1	The methods used resulted in a new best found solution
σ_2	The methods used resulted in a solution that has not been accepted before, and which is better than the current solution.
σ_3	The methods used resulted in a solution that has not been accepted before, and which is worse than the current solution.

We make a distinction between the second and the third rule, because we do not only want to reward methods that are able to find a better solution. We also want to reward methods that are able to diversify the search and visit solutions that have not been visited before. To determine whether a solution has been visited before, we make use of a method which determines whether two solutions are equal. The method treats two solutions as the same solution whenever the objective value is equal and for every route in one solution, we can find a route in the other solution that services exactly the same requests.

If a segment is complete, we calculate new weights for the methods based on the scores from the last segment. In the first segment every method is given a weight of 1. Then, at the end of segment j, we calculate the weight of destroy method d in segment $j+1$ as

$$\rho_{d,j+1} = (1-r)\rho_{d,j} + r\frac{\pi_d}{\theta_d}. \tag{36}$$

Here, π_d is the score of the method obtained during the last segment, while θ_d is the number of times the method has been applied. Parameter r is a *reaction factor* which controls how sensitive the updating procedure is to the effectiveness of the method in the last segment. The updating procedure for the repair weights is similar. As a consequence

of this weight updating procedure, the probabilities for choosing every destroy- and repair method within each segment stay the same until the end of this segment. Every segment therefore represents a period of learning, during which the methods are given scores based on their performance, and after which the weights for the next segment are updated accordingly.

An overview of the ALNS algorithm is given in Algorithm 1.

Algorithm 1 ALNS algorithm

Get initial solution $\mathcal{X}$ using construction heuristic
Set best solution $\mathcal{X}^* = \mathcal{X}$
Set initial weights to 1 and initial scores to 0
$i \leftarrow 1$
while stopping criterion not met **do**
    Select destroy method $d \in \Omega^-$ based on ρ^-
    Select repair method $r \in \Omega^+$ based on ρ^+
    $\mathcal{X}' \leftarrow d(\mathcal{X})$
    $\mathcal{X}'' \leftarrow r(\mathcal{X}')$
    tuneSolution$(\mathcal{X}'')$
    if *containsConflicts*$(\mathcal{X}'')$ **then**
        resolveConflicts$(\mathcal{X}'')$
    end if
    if conflicts cannot be resolved in $\mathcal{X}''$ **then**
        continue while
    end if
    if $f(\mathcal{X}'') < f(\mathcal{X})$ **then**
        $\mathcal{X} \leftarrow \mathcal{X}''$
    else if SA condition satisfied **then**
        $\mathcal{X} \leftarrow \mathcal{X}''$
    end if
    if $f(\mathcal{X}) < f(\mathcal{X}^*)$ **then**
        $\mathcal{X}^* \leftarrow \mathcal{X}$
    end if
    if $i \equiv 0 (mod\ n_{seg})$ **then**
        Update weights and set all scores to 0
    end if
    Update T and scores
    $i \leftarrow i + 1$
end while

4.1.2 Minimal recharge assumption

In Section 5, we will compare a partial recharge- to a full recharge policy. Using the latter, the amount recharged at a CS always equals the difference between maximum capacity and the remaining battery level upon arrival. However, when using a partial recharge policy, any nonnegative amount of energy can be recharged, given that maximum capacity is not exceeded. Using this policy in our ALNS framework, the question arises whether or not some insertion is feasible or not. That is, given that we make an insertion of a request in some route, is the resulting route still feasible with respect to all the conditions proposed in Section 3.1? To answer this question under a partial recharge policy, we

make an assumption on the amount recharged. To be specific, we assume that every EV uses a minimal recharge policy. Consequently, the amount recharged at a CS equals the difference between the energy needed to reach the next CS or the depot and the remaining energy upon arrival at this CS, provided that this difference is nonnegative. If we disregard the superscript for some path h from the PBF in Section 3.3, then the amount recharged by EV k at CS f equals

$$\zeta_f^k = \max(0, \gamma_{f,next}^k - \beta_f^k). \tag{37}$$

Here, $\gamma_{f,next}^k$ is the energy needed to reach the next CS, or the depot in case no CS succeeds the current CS f somewhere on the route.

Under the assumption of a minimal recharge policy, the recharged amounts for a given solution $\mathcal{X}$ are known and we know if the insertion of a request on a route is feasible or not.

4.2 Construction heuristic

The construction heuristic is based on the one proposed by Keskin and Çatay (2016) for the E-VRPTW. The heuristic is described in Algorithm 4 in Appendix B. First, we start a new route with the request that is closest to the depot, determined by the distance measure $\mathcal{D}_{i,n+i} = \{d_{0,i} + d_{i,n+i} + d_{n+i,2n+1}\}$, for a request with $i \in P$ and $n + i \in D$. Then, if U is the set of pickups corresponding to unhandled requests, we calculate the value Δf_{ir}^{pq} for every pickup $i \in U$. This represents the change in distance of the current route r when we insert pickup i at position p and its delivery $n + i$ at position q. We only consider insertions for which $p < q$, such that a delivery is always inserted after its pickup. We now try to insert the request corresponding to the lowest insertion cost.

If this insertion is not energy feasible, we apply the method *insertCS*. It determines the first customer at which the EV from the current route arrives with a negative battery level. For all arcs preceding this first violated customer, this method adds the closest CS on every arc. The procedure keeps on adding the closest CS to every arc until r becomes energy feasible or until all arcs contain a CS. If all arcs that precede the first violated customer contain a CS, but r is not energy feasible, the method inserts a second closest CS on every arc. Consequently, it may occur that *insertCS* does not return an energy feasible route, even though all arcs preceding the first violated customer contain two CSs. In this case, the insertion is marked as an infeasible insertion.

In case that the modified route is energy feasible, the method checks if this modified route is feasible with respect to cargo and time window restrictions. If it is, we update the current route to be this modified route. If no request can be inserted into the current route, we start a new route with the unhandled request j having the smallest value for

$\mathcal{D}_{j,n+j}$. When all unhandled pickups corresponding to unhandled requests are inserted in some route, we check for conflicts at CSs and try to resolve them. This means that similar to Ding, Batta, and Kwon (2015), the CS capacity constraints are first relaxed, after which in an existing set of routes any conflicts are resolved. In an attempt to resolve the conflicts, we make use of the method *resolveConflicts*, which is explained in more detail in Section 4.5.

4.3 Destroy methods

In general, every destroy method $d \in \Omega^-$ removes some requests from an existing solution. In order to remove a request from an existing route, both the customers from the pickup and the delivery corresponding to this request are removed from their positions and the remaining customers from this route are repositioned accordingly. Every destroy method which is not random in nature, is given an element of randomness. The degree of randomness is controlled by parameter p. The randomness is added to deal with the local behaviour of the methods and diversify the search for a global minimum.

4.3.1 Request removal

If a request removal method is executed, we choose q as a random, integer number of requests between 1 and $|P|\xi_{request}$, where $|P|$ is the total number of requests and $\xi_{request}$ is a parameter that controls the degree of solution destruction.

The first destroy method, *Random Removal* (RR), simply selects q requests and removes them from the solution. These requests may be from the same route, or from different routes.

In order to remove requests that contribute to a high objective value, we incorporate *Worst Removal* (WR). In this method, we determine for each request in the solution the decrease in objective value if we were to remove this request from its route. Then, we order the requests in a list starting with the requests causing the largest decrease in case it were to be removed. We remove the requests from the list from their routes until q requests have been removed. To add randomness to this method, we skip an element from this list with probability p. The idea behind this method is that these removed requests may be inserted at another position (and maybe in another route) at a lower cost.

Late Arrival Removal (LAR) is similar to WR, however instead of the decrease in objective value we determine for each request with $i \in P$ and $n+i \in D$ a total lateness measure. This measure is defined by

$$L_i = (\tau_i - e_i) + (\tau_{n+i} - e_{n+i}), \tag{38}$$

and is a measure of how late the start of service times at a vertices i and $n+i$, τ_i respectively τ_{n+i}, are compared to the earliest possible start of service times. Here, τ_i and τ_{n+i} are defined by the recharge policy used, as explained in Section 4.1.2. By construction of a solution, τ_i is always at least as high as e_i for any vertex i and therefore L_i is always nonnegative. In decreasing order of L_i, we remove q requests from the solution, where randomness is introduced by parameter p in a similar manner as in WR. The idea is that late start of service times restrict the remaining part of a route, thereby limiting the insertion possibilities in terms of time windows. A removed request with a high lateness may be inserted at another position (or another route) more efficiently in terms of time.

Related Removal (RLR) determines for every pair of distinct requests a dissimilarity score. Given distinct requests with r and $s \in P$, this score is defined by

$$R(r,s) = \phi(d_{r,s} + (d_{n+r,n+s})) + \chi(|\tau_r - \tau_s| + |\tau_{n+r} - \tau_{n+s}|) + \psi|q_r - q_s|, \tag{39}$$

and indicates how dissimilar the requests r and s are. A high value of $R(r,s)$ indicates high dissimilarity. d_{ij} is the distance between any two locations i and j and q_i is the demand of location i. Using parameters ϕ, χ and ψ, the measures of dissimilarity are weighted. Similar to Ropke and Pisinger (2006), we decide to normalize the values d_{ij}, τ_i and q_i. We therefore determine the maximum and minimum values for these parameters among all i and j a priori and then scale the parameter value at hand. For example, for any two vertices $i \in V_0$ and $j \in V_{2n+1}$, if we know the distance d_{ij} between them, we calculate the normalized distance

$$z_{ij} = \frac{d_{ij} - \min\limits_{i \in V_0, j \in V_{2n+1}} \{d_{ij}\}}{\max\limits_{i \in V_0, j \in V_{2n+1}} \{d_{ij}\} - \min\limits_{i \in V_0, j \in V_{2n+1}} \{d_{ij}\}}, \tag{40}$$

which always lies between 0 and 1. This way, a dissimilarity measure cannot have a bigger effect on the total dissimilarity between two requests, simply because of its scale. As an example, if one had distances in kilometers but were to change them to meters, the importance of the distance measure would increase even though the similarity remains the same.

The method starts by removing a random request, and then determines the $q-1$ requests that are most similar to this randomly chosen request. In total, q requests are removed from the solution.

4.3.2 Route removal

If a route removal method is executed, we choose q as a random, integer number of routes. This number depends on the number of routes in the current solution and is determined between 1 route and the integer number of routes resulting from rounding down $100\xi_{route}\%$ of the total number of routes. Similar to the request removal methods, parameter ξ_{route} is a parameter that controls the degree of solution destruction.

Random Route Removal (RRR) simply selects q routes and removes the customers from these routes. The selected routes will therefore have an empty route when RRR is applied. The main idea behind this random method is to diversify the search for a global minimum.

Greedy Route Removal (GRR) selects q routes, from which the customers should be removed, in a greedy manner. The existing routes are sorted in non-decreasing order of number of customers serviced, or equivalently number of requests serviced. The q routes are selected from this sorted list and randomness is introduced by skipping the selection of a route with probability p. The idea behind this method is to distribute the requests from shorter routes into longer routes such that the number of EVs used may be reduced.

4.3.3 Hybrid CS and request removal

We design an ad-hoc method called *Crowded Station Partial Route Removal* (CSPRR). It determines the most crowded CSs in terms of number of arrivals during the entire planning horizon $[0,\ l_{2n+1}]$. Starting with the arrivals that occur at the most crowded CS, it selects up to q arrivals. If q arrivals are selected from the most crowded CSs, the method destroys a part of the route that these arrivals are in. The idea is that destroying part of the route will lower the energy required on the route, so that removing the crowded CS arrival(s) from the route might not lead to an energy infeasible subroute. The heuristic therefore focuses more on intensification instead of diversification of the solution space. In case that we have selected multiple crowded arrivals from the same route, the method determines the latest arrival in this route. Then, it finds the first customer that succeeds the CS of the latest arrival on the route and removes this customer and also its corresponding pickup/delivery from the route. If no more customers succeed this CS on the route, it removes the first customer preceding this CS on the route and also removes its corresponding pickup/delivery.

The rationale behind this method is that crowded CSs may lead to conflicts. The method *resolveConflicts* aims at resolving these conflicts, but is myopic in the sense that it is applied only once at the very end of every iteration. To diversify the search for conflict-free routes, we decide to tackle potential conflicts already during the selection of a candidate solution by destroying some routes that contain a crowded CS arrival.

The value for q is determined in the same way as done for route removal methods.

4.4 Repair methods

In general, every repair method $r \in \Omega^+$ inserts a number of requests into a partially destroyed solution until all requests have been inserted in some route and all routes from the solution are feasible.

The first repair method, *Greedy Insertion* (GI), is similar to the construction heuristic. Given a partial solution $\mathcal{X}$ and a list of unhandled requests, the method calculates the value Δf_{ir}^{pq} for every unhandled pickup $i \in U$ corresponding to an unhandled request and for every existing route $r \in \mathcal{X}$. That is, for every unhandled request we consider the possibility of inserting this request on every existing route and for all possible positions p and q. Then, in increasing order of Δf_{ir}^{pq}, we try to make the insertion. If no request can be inserted, then we start a new, empty route. The key difference with the construction heuristic is that in GI, we look at possible insertions in all existing routes, instead of only the current route.

Regret-2 Insertion (R2I) is similar to GI, however instead of calculating the value Δf_{ir}^{pq} for every unhandled pickup, we calculate so-called *regret values*. We define the regret value for a request as the difference between the second lowest value for Δf_{ir}^{pq} and the lowest value for Δf_{ir}^{pq} among all existing routes. Following the notation from Ropke and Pisinger (2006), this is the definition of the *regret-2* value and therefore we call this method the *Regret-2 Insertion*. The idea is that, for a given request, if a second best insertion is much worse (higher) than a best insertion, we will regret not making this insertion now. Therefore we choose the biggest regret value among unhandled requests and try to make this insertion in the same manner as done in GI.

Time Based Insertion (TBI) is similar to GI, however instead of calculating the value Δf_{ir}^{pq} for every unhandled pickup, we calculate a time efficiency measure. This time measure is defined as the difference in total route duration before and after insertion of an unhandled request. Similar to GI, we evaluate the possibility of inserting the pickups and deliveries of all unhandled requests at all possible positions p and q. After all, different insertion positions in the same route imply different total route durations because of different distances and amounts recharged. Starting with the insertions that increase the route duration the least, we insert requests until no more requests are unhandled. The idea behind this method is to make the existing routes more efficient in terms of time, thereby allowing more potential for route creation and avoiding the need to start a new route. Also, more efficient routes potentially allow more flexibility for *resolveConflicts* to shift arrivals.

4.5 Detect and resolve conflicts

A conflict occurs at a CS whenever an EV is charging at this CS and another EV arrives to start its recharge operation. To determine whether any conflicts exist, the method

containsConflicts first sorts all the arrivals at any CS over the entire planning horizon in increasing order of start of service time. Then, for every arrival in the sorted list, it checks whether there is a conflict with this arrival and any of the arrivals that precede this arrival in the list. In particular, there is a conflict whenever the two arrivals occur at the same CS and the start of service of the current arrival is less than the end of service of the previous arrival.

The method *resolveConflicts* aims to resolve the conflicts from a solution. For every CS, this method finds all the *conflict arrivals*. That is, it finds all the arrivals that have at least one conflict with another arrival. The main idea behind this method is to concatenate the conflict arrivals at this CS in the order of their start of service times. Since EVs are allowed to wait at a CS before starting their recharging operation, concatenation may be a way to resolve conflicts whenever shifting a conflict arrival further in time does not give any time violations at subsequent locations on the route.

The method starts by sorting the arrivals at every CS based on their start of service times. The idea is to detect clusters of conflict arrivals, and to concatenate the conflict arrivals from every cluster to each other in order of their start of service times. Here, the clusters between conflict arrivals adhere to the following law: if arrival A is in conflict with arrival B and B is in conflict with arrival C, then arrivals A, B and C are all contained in the same cluster. If clusters are identified, the conflict arrivals from every cluster are concatenated to the earliest conflict arrival from this group one at a time, starting with the second earliest arrival. As a result, for a cluster with three conflict arrivals, the second arrival is shifted right after the first, and the third is shifted right after the new position of the second arrival. If an arrival cannot be shifted, it is placed in a special list, and the method tries to concatenate the next earliest arrival. This way, the method tries to create a conflict-free schedule for every CS separately, while all the arrivals from every CS that could not be shifted are placed in the same special list. The special list may thus contain arrivals occuring on different CSs.

There are two reasons an arrival cannot be shifted, namely:

(i) shifting the arrival will give a time violation at one of the subsequent locations on the route that this arrival is in;

(ii) shifting the arrival will give a conflict with the first conflict-free arrival or first arrival from the next cluster that succeeds the current arrival.

Reason (ii) limits the shifting flexibility of this method in case that there exists a conflict-free arrival that succeeds a cluster or in case that there exists a cluster that succeeds the current cluster. We do this to keep the run-time of the algorithm low. As a result, our method focuses only on resolving within-cluster conflicts and it does not consider the possibility of shifting complete clusters in time such that more flexibility is created in some preceding cluster.

Furthermore, we implement the restriction that if some conflict arrival cannot be shifted and is placed in the special list, then also any other conflict arrival that is positioned in the same route is placed in the special list.

An example is shown in Figure 1, in which 5 different EVs all make at least one stop at CS f during their routes. For simplicity, the arrivals are denoted by their time intervals and the corresponding routes are hidden. These time intervals are defined by the start of service and end of service times in case the conflict-free constraints are ignored. We observe that the first cluster of conflict arrivals contains the arrivals [200, 210] and [205, 230] and the second cluster contains the arrivals [310, 350], [315, 320] and [340, 370]. It is worthwhile to note that conflict arrivals [315, 320] and [340, 370] are from the same route.

The method fixes arrival [200, 210] as it is the first arrival in that cluster. It then tries to shift arrival [205, 230] to time 210. It checks whether time violations occur at subsequent locations on the route of EV 2. Also, it checks whether the first conflict-free arrival that succeeds the cluster from [205, 230], which is arrival [250, 290], does not give any problems based on reason (ii). Given that we shift the arrival of [205, 230] to 210, EV 2 will finish its recharging operation at time $210 + (230 - 205) = 235$. As EV 3 does not arrive until time 250, we do not have a conflict with this EV if we shift [205, 230]. Now the method moves on to the next cluster and tries to shift arrival [315, 320] to time 350, without shifting the first arrival [310, 350] from this cluster. If the shift of arrival [315, 320] to time 350 is not possible for any of the two reasons, then the shift of arrival [340, 370] also becomes impossible due to our extra restriction. In that case, both arrivals [315, 320] and [340, 370] are placed in the special list.

In the example, in theory we could consider the possibility of shifting arrival [250, 290] and the second cluster accordingly, such that more shifting flexibility is created to resolve the conflicts in the first cluster. However, our method does not consider this possibility due to reason (ii).

If all CSs are handled, the method considers the special list, which contains the arrivals that could not be shifted for any of the two reasons. In particular, we try to reposition every arrival from this list to another, closest CS, given that this respositioning is feasible with respect to energy, cargo and time windows. Also, no conflict may occur with the arrivals that currently take place at this other CS. That is, the method only considers a respositioning to a CS that is free at the time interval corresponding to the arrival. If any of the arrivals from the list cannot be repositioned, then the method concludes that the solution cannot be made conflict-free.

Ding, Batta, and Kwon (2015) have also used a two-stage approach of first relaxing the capacity constraints on the CSs and then resolving any conflicts. In their second stage, the authors do not concatenate the arrivals based on their start of service times but instead they concatenate the arrivals based on the makespans of the routes to which

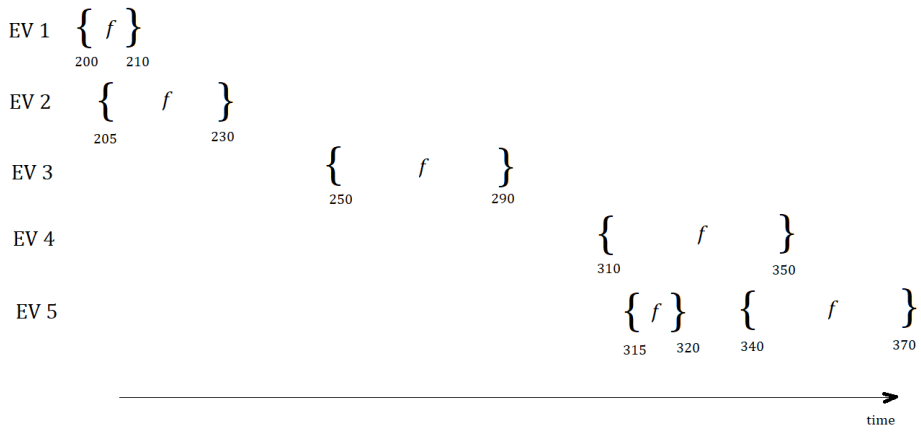

Figure 1: Example input for method *resolveConflicts* with a focus on CS f at which 5 different EVs make one or more stops.

these arrivals belong. Starting with the arrival on the route having the largest makespan, the conflicting arrivals are concatenated if time windows allow this. In contrast to our approach, the authors also adjust the charging times by increasing the recharged amount, while in our method the recharged amount is always kept to a minimum under a partial recharge policy and always kept to a maximum under a full recharge policy.

5 Numerical studies

In this section we present numerical results of the proposed ALNS heuristic applied to the PDPTW-HEVC. In order to do so, we will first present a number of test instances for the problem, as discussed in Section 5.1. Then, we will discuss important points of implementation for the comparison between our exact method and the ALNS in Section 5.2. Next, we will discuss the parameter tuning process in Section 5.3. Finally, we present our results in Section 5.4. All experiments were run on a system using an Intel Core i5 processor at 1.60 GHz and 8 GB of RAM, operating on Windows 10 Home. We implement the exact formulation in Java and use solver CPLEX version 12.6 to obtain an optimal solution.

5.1 Test instances

The test instances are based on the instances provided by Goeke (2019) for the PDPTW-EV. These instances consist of 36 small-sized instances with anywhere between 3 and 9 requests, and 56 large-sized instances with up to 50 requests. They are based on the instances provided by Schneider, Stenger, and Goeke (2014) for the E-VRPTW, which are in turn based on the well-known benchmark instances provided by Solomon (1987). The instances vary in their request distribution, i.e. the customers from the requests might be clustered (c), randomly distributed (r) or a combination of both (rc). Also, the instances have either a shorter (1) or longer (2) scheduling horizon. Finally, every large-sized instance is indicated with l.

We modify the total of 92 instances such that they represent PDPTW-HEVC instances, by replacing the information on a homogeneous fleet with information on a heterogeneous fleet. In particular, we construct for every instance a $|K|$ x 7 matrix where every row corresponds to an EV and the 7 columns contain information on type, battery capacity, freight capacity, battery consumption rate, inverse recharge rate, average velocity and acquisition cost.

Similar to Hiermann et al. (2016), we make use of the vehicle types defined by Liu and Shen (1999), which differ in their freight capacity and acquisition cost. Generally, bigger vehicles can carry more freight but also have a higher acquisition cost. On top of a proportionally higher acquisition cost for every type, Liu and Shen (1999) have defined three levels of acquisition costs: small, medium and large (Belfiore and Fávero (2007)). However, we decide to make no distinction between these different levels and incorporate only the medium level acquisition costs in our matrix, which already define an increasing relation between the acquisition costs and the EV types.

Similar to the instances from Goeke (2019), we assume that the average speed and the battery consumption rate always equal 1.

In line with Hiermann et al. (2016), we use the battery capacity of the unmodified

instances from Goeke (2019) as base value and scale this value up or down based on the EV type (and thus the EV size) at hand. We do this to model the assumption that bigger EVs are also capable of carrying a bigger battery with a larger capacity. The scaling itself is also done in a way similar to Hiermann et al. (2016) and formula (41) is equivalent to the formula used by the authors. For a moment, we index the vehicle types by $\mathcal{T} = 1, 2, ..,$ instead of using letters $A, B, ..,$ as in Liu and Shen (1999). We do maintain the correct order, so EV type $\mathcal{T} = 1$ corresponds to EV type A from Liu and Shen (1999) and so on. If an instance has a total number of $|\mathcal{V}|$ vehicle types and a base value battery capacity of Q_{base}, and if we omit for simplicity the index k from Section 3.1, then the battery capacity of an EV of type $\mathcal{T} = 1, 2, ..,$ is given by

$$Q_{\mathcal{T}} = Q_{base}(1.0 + 0.1\mathcal{T} - 0.1|\mathcal{V}|/2). \tag{41}$$

In case $\mathcal{T}$ is smaller than $|\mathcal{V}|/2$, the expression $0.1\mathcal{T} - 0.1|\mathcal{V}|/2$ is negative and the base value is scaled down. In case $\mathcal{T}$ equals $|\mathcal{V}|/2$, we leave the base value as it is for the current EV type. Finally, in case $\mathcal{T}$ is greater than $|\mathcal{V}|/2$, the base value is scaled up. Therefore, we can think of this formula as a way to assign the base value battery capacity to the EV type *in the middle* and to scale up or down the battery capacities of all other EV types based on their position relative to the *middle* EV type (assuming the types are ordered in a list).

All that remains is to decide on the total number of EVs $|K|$. In this research we assume that the number of EVs from every type is unlimited. This means that when constructing the instances, we generate for every instance a sufficient number of $|P|$ EVs from every type, i.e. a total of $|P||\mathcal{V}|$ EVs. Here, $|P|$ is the number of requests. Note that this number is sufficient because every request must be serviced by exactly one EV so that no fleet possibilities are eliminated.

It turns out that for our exact method, this strategy of generating $|P||\mathcal{V}|$ EVs can lead to implementation issues, as we will discuss in Section 5.2.

During preliminary testing, we found that our exact method could not find a feasible solution for instance *c101C6* because the number of EVs, n^*, was too low. Therefore, we increased the battery capacities of all EVs by 10% for this instance. Also, we increased the battery capacities of some large instances in order to prevent these instances to be infeasible. For more details, see Section 5.4.2.

5.2 Implementation issues influencing the comparison

In Section 3.3.5 we have already touched upon an important implementation issue for the exact method. The constraint sets that avoid conflicts at CSs can grow large due to the fact that they are defined for all pairs of distinct EVs, for all CSs, and for all pairs of

distinct RPs that contain this CS.

During preliminary testing, we found that generating a sufficient number of $|P||\mathcal{V}|$ EVs for every instance led to the fact that none of the instances could be solved by our exact method within reasonable time or without running out of memory. In order to tackle this issue, we decide to keep the number of EVs as low as possible. To this end, we decide to give a priori information on the total number of EVs available before solving an instance with our exact method. In particular, we come up with a strategy that attempts to prevent our exact method from running out of time or memory, such that it can still be used as a benchmark to assess the performance of our ALNS on the small instances. The key point is that the total number of EVs is restricted to some number $n^* < |P||\mathcal{V}|$.

Our strategy uses the results from Goeke (2019) for the PDPTW-EV. In particular, if we want to compare our exact method to the ALNS for some small instance, let n^* be the number of EVs that Goeke (2019) reported in the final solution of the same instance. We decrease the number of EVs for this instance from $|P||\mathcal{V}|$ to n^*. Given the total number of EV types as defined by Liu and Shen (1999), we now solve our exact method repeatedly, each time with a different fleet composition. This way, the exact optimal solution will be found, under the assumption that the fleet consists of n^* EVs. As an example, an instance with $n^* = 2$ and 3 EV types, A, B and C, will have 6 different possible fleet combinations: AA, AB, AC, BB, BC, and CC. Solving our exact method for all these different fleet combinations, we find the optimal solution for a fleet of size 2.

Now the goal is to compare this optimal solution to the one obtained by the ALNS. It is important to note that the heuristic will generally evaluate solutions that consist of more than n^* EVs. We therefore decide not to give the same a priori information on the number of EVs while running the heuristic. The aim is now to evaluate the performance of the ALNS, which is able to choose from a number of $|P||\mathcal{V}|$ EVs, by comparing it to the exact optimal solution which consists of at most n^* EVs. Prior to our experiments, we already note that this strategy is limited in the sense that ALNS might find a better solution because it is allowed to select more than n^* EVs.

5.3 Parameter tuning

The ALNS algorithm contains a number of parameters. The initial values for these parameters are chosen based on existing literature, resulting in the following parameter vector: (w, c, n_{seg}, σ_1, σ_2, σ_3, r, p, $\xi_{request}$, ξ_{route}, ϕ, χ, ψ) = (0.05, 0.99975, 100, 33, 9, 13, 0.1, 0.5, 0.4, 0.3, 9, 3, 2). In particular, parameters w, c, n_{seg}, σ_1, σ_2, σ_3, r, $\xi_{request}$, ϕ, χ and ψ are based on the values from Ropke and Pisinger (2006), while the value for p is chosen ad hoc. The initial value for ξ_{route} is set to the tuned value by Keskin and Çatay (2016).

The initial parameter setting is then improved to boost the algorithm's performance. For the improvement phase, we assume a partial recharge policy. Also, we determine a set

of representative test instances. Given the different categories of instances as presented in the beginning of Section 5.1, we build a set of representative instances consisting of *c101C6*, *c202C12*, *lc101*, *lc201*, *lr101*, *lr201*, *lrc101*, *lrc201*, *r102C12*, *r201C10*, *rc102C12* and *rc201C10*. Then, we start the improvement phase by determining a good value for the first parameter w. We allow it to take on a number of values while all other parameters are kept fixed. For each different value, leading to a different parameters setting, we run the heuristic on our set of representative instances three times. The value that gives the best average behaviour, in terms of $f(\mathcal{X})$ achieved within the same time limit, is chosen. Now, the second parameter c is tuned in a similar manner, while w is kept at its optimal value. We continue in this way until all parameters are tuned. Finally, we obtain the following tuned parameter vector: $(w,\ c,\ n_{seg},\ \sigma_1,\ \sigma_2,\ \sigma_3,\ r,\ p,\ \xi_{request},\ \xi_{route},\ \phi,\ \chi,\ \psi)$ = (0.075, 0.99975, 100, 23, 4, 19, 0.02, 0.75, 0.2, 0.3, 3, 3, 8). The evaluated parameter values, together with the initial and optimal parameter value are presented in Table 7 in Appendix C.

5.4 Results

In every experiment we assume that whenever a new route is created, we select among non-occupied EVs the one with the lowest acquisition cost, to serve this route.

5.4.1 ALNS compared to exact formulation

We implement the comparison between the exact method and the ALNS only for a subset of the small-sized instances. That is, we only consider the instances for which, given the number of EV types and the value of n^*, the number of possible fleet compositions does not exceed 15. This value is chosen ad hoc, leaving us with 24 out of 36 small-sized instances to consider. Having tried different comparison strategies on the remaining 12 instances, it turns out that a fair comparison was not possible as the exact method always ran out of memory regardless of the comparison strategy. In order to still make use of these 12 instances, we decide to add them to the set of large-sized instances to be used in Section 5.4.2.

We solve the 24 small-sized instances using CPLEX restricting the run-time to 5 minutes. Also, we perform three runs of the ALNS on these instances, where the run-time is also restricted to 5 minutes (excluding time needed to find an initial solution). We have purposely set the time limit for the exact method not too high: after all, in the worst case the same instance needed to be solved 15 times (once for every possible fleet composition). We report the best found solution. Furthermore, we decide not to implement a stopping criterion in terms of a maximum number of iterations without improvement, as preliminary testing showed that this led to premature convergence of the algorithm giving sub-optimal solutions. We assume that all EVs use a partial recharge

Table 3: Comparison between results on small instances obtained by exact method and ALNS.

		MIP					ALNSPR				Comparison	
Instance	n^*	fleet	$f(\mathcal{X})$	$\mathcal{F}$	$f_{lb}(\mathcal{X})$	t	fleet	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$	Δ_f	% off
c101C6	2	A^2	384.44	3		2103	A^2	384.44	3	0.22	0	0
c103C6	1	A	235.37	3		286	A	235.37	3	0.01	0	0
c206C6	1	A	450.28	4		312	A^2	645.81	3	0.0	195.53	43.42
c208C6	1	A	364.34	2		297	A	364.34	2	0.17	0	0
r104C6	2	B^2	223.62	3		4155	B^2	223.62	3	0.0	0	0
r105C6	2	AC	204.31	3		1138	AC	204.31	3	0.0	0	0
r202C6	1	A	232.65	2		331	A	232.65	2	0.09	0	0
r203C6	1	A	269.06	3		313	A	269.06	3	0.07	0	0
rc105C6	2	B^2	293.77	3		1697	B^2	301.30	3	0.18	7.53	2.56
rc108C6	2	B^2	313.93	3		1443	AB^2	399.64	3	0.0	85.71	27.30
rc204C6	1	A	218.09	4		490	A	226.74	4	0.32	8.65	3.97
rc208C6	1	A	204.89	3		300	A	204.89	3	0.01	0	0
c104C10	2	A^2	$+\infty$		371.34	TL	A^2	489.58	4	0.28	-	-
c205C10	2	AB	751.75	2		40526	AB	751.75	2	0.61	0	0
r201C10	1	A	354.72	5		1123	A^2	434.16	4	0.22	79.44	22.40
r203C10	1	B	358.21	4		3662	B	372.68	4	0.48	14.47	4.04
rc201C10	1	C	522.86	6		844	A^2	450.58	7	0.09	-72.28	-13.82
c202C12	1	A	504.06	5		1071	A^2	694.43	4	0.62	190.37	37.77
r103C12	2	BC	$+\infty$		206.76	TL	B^3	307.70	3	0.0	-	-
c202C16	2	A^2	$+\infty$		673.04	TL	A^3	1121.52	7	1.51	-	-
c208C16	2	A^2	$+\infty$		667.63	TL	A^2	755.95	4	1.31	-	-
r202C16	2	A^2	$+\infty$		448.17	TL	B^2	788.28	9	0.99	-	-
r209C16	1	B	481.04	7		304954	A	530.46	10	0.97	49.42	10.27
rc204C16	1	B	$+\infty$		337.12	TL	A^2	490.72	7	0.59	-	-
Average			353.74	3.71		20280.28		384.79	3.67	0.23	31.05	8.78

policy.

The results on the small instances obtained by our exact method and the ALNS are presented in Table 3. We indicate with ALNSPR the ALNS with a partial recharge policy implemented. We report the instance (Instance) and the final number of EVs as obtained by Goeke (2019) (n^*). For both the exact method (MIP) and the ALNS we also report the fleet composition (fleet), objective value ($f(\mathcal{X})$) and the number of CSs visited ($\mathcal{F}$). The trivial upper bound $+\infty$ was reported for instances for which the exact method could not find a valid upper bound within the time limit. Additionally for the MIP, we report a lower bound on the optimal objective ($f_{lb}(\mathcal{X})$) in case the time limit is reached and we report the run-time in milliseconds (t). If the time limit was reached, we indicate this with TL. Additionally for the ALNS, we report the average number of conflicts during the search ($\mathcal{C}$), defined as the total number of conflicts found divided by the total number of iterations. Finally, for the instances that could be solved to optimality we implement a comparison between the two methods by reporting the difference in objective (Δ_f) and how many percent the ALNS is off from the solution found by our exact method (% off).

In the last row of our table, some relevant averages are reported, only taking into calculation the instances for which optimal solutions were obtained within the time limit.

Some important notions should be made. For some instances our exact method was not able to find an optimal solution for any of the fleet compositions within the time limit. In this case, we selected the optimal fleet composition as the one that had found the lowest lower bound at the moment the time limit was reached. Next to this, for some instances our exact method was able to find optimal solutions for some of the fleet compositions, but not for others. In that case, we selected the optimal fleet composition as the one that had found the lowest optimal solution.

From the table, we see that our exact method was able to find optimal solutions for 18

out of 24 small instances. Our ALNS was able to match these optimal results on only 9 of the instances. On average, ALNS was 8.78% off from the objective obtained by the MIP. On 14 instances, ALNS found the same optimal fleet composition as the one provided by our exact method. The number of CSs visited in ALNS solutions was on average only 0.04 lower. The average number of conflicts observed per ALNS iteration was 0.23, i.e. about once every four iterations a conflict was observed which needed to be resolved.

For the instances that could not be solved to optimality within the time limit, the objective obtained by ALNS was on average 46.21% off from the lower bound obtained by the exact method. However, this gap will decrease if we give the exact method more time because the lower bounds will increase.

On instance *rc201C10* the solution obtained by the ALNS was actually better than the solution obtained by the MIP due to the fact that it was more beneficial to use more EVs with a lower acquisition cost, but the MIP was restricted to the use of $n^* = 1$ EV.

Instances *rc108C6, rc201C10, r103C12, r209C16* and *rc204C16* show us that for the PDPTW-HEVC it is sometimes more beneficial to use more expensive EVs due to the trade-off between total route distances and total acquisition cost. That is, on these instances the strategy of always selecting the EV with the lowest acquisition cost among non-occupied EVs is not sufficient to find the optimal set of routes.

We observed that the optimal solution to the PDPTW-HEVC is very sensitive to certain parameter adjustments in the ALNS. In particular, some instances require a large degree of destruction, while others require a smaller degree of destruction in order to find the optimal solution. On instances *c206C6*, *rc108C6*, *r203C10*, *c202C12* and *r209C16*, we quickly found (near-)optimal solutions matching the exact results when we would simply increase the values of $\xi_{request}$ and ξ_{route}.

Finally, we argue that inserting up to two CSs between every two customers on a route, as explained in Section 4.2, should have given ALNS the same amount of flexibility as the MIP to find the optimal solution. That is, we found that the optimal routes from our exact method never contained more than 2 CSs in between customers on a route.

5.4.2 Partial- compared to full recharge policy

We now turn our attention to the comparison between a partial- and full recharge policy. The former has been proven in existing research to be superior to the latter, and the aim of this section is to find out whether this notion also holds for the PDPTW-HEVC. The experiments here were performed on the 12 small-instances that were not suitable for the comparison in Section 5.4.1, together with the 56 large-sized instances.

We solve the 68 instances by performing three runs of ALNS, each time restricting the run-time to 5 minutes (excluding time needed to find an initial solution). We report the best found solution. Again, we decide not to implement a stopping criterion in terms of a maximum number of iterations without improvement. We indicate with ALNSFR the

Table 4: Comparison between results obtained by partial- versus full recharge policy

	ALNSPR				ALNSFR				Comparison	
Instance	fleet	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$	fleet	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$	Δ_f	% off
rc108C10	ABD	529.62	2	0	ABD	529.62	2	0	0	0
rc205C10	A^2	522.84	7	0.43	A^2	524.53	9	0.97	1.69	0.32
c101C12	A^3	598.19	5	0.49	A^3	598.4	5	0.04	0.21	0.04
r102C12	ABC^2	374.76	2	0	ABC^2	374.76	2	0	0	0
rc102C12	A^4B	544.58	1	0	A^4B	544.58	1	0.01	0	0
c103C16	A^3	620.11	5	0.48	A^3	679.66	7	1.21	59.55	9.60
c106C16	A^3	556.15	4	2.07	A^3	589.58	4	1.87	33.43	6.01
r105C16	A^2BD^2	525.75	4	0.04	A^3BD	515.32	6	0.13	-10.43	-1.98
rc103C16	A^3BD	686.82	2	0.21	A^3BD	686.82	2	0.65	0	0
rc108C16	A^3B^2	593.04	4	0.69	A^3B^2	593.04	4	1.43	0	0
rc202C16	AC	730.56	10	1.55	AC	715.01	10	1.01	-15.55	-2.13
r102C18	A^4B^3	574.89	6	0.12	AB^3C^2	535.55	5	0.83	-39.34	-6.84
lc101	A^{12}	1843.46	10	1.02	A^8B^2	1859.33	9	3.05	15.87	0.86
lc102	A^{13}	1906.99	10	1.81	A^{13}	1885.75	11	4.65	-21.24	-1.11
lc103	A^{12}	1795.03	10	2.73	A^{12}	1826.42	10	3.42	31.39	1.75
lc104	A^{11}	1718.09	10	2.85	A^{12}	1859	11	4.72	140.91	8.20
lc105	A^{12}	1860.65	12	1.4	A^{13}	1953.14	13	2.29	92.49	4.97
lc106	A^{12}	1765.21	9	1.37	A^{12}	1872.85	12	3.28	107.64	6.10
lc107	A^{11}	1650.75	9	1.5	A^{12}	1769.61	7	2.2	118.86	7.20
lc108	A^{12}	1754.83	6	1.86	A^{11}	1633.92	6	2.3	-120.91	-6.89
lc109	A^{10}	1494.82	9	2.04	A^{12}	1782.49	12	4.31	287.67	19.24
lc201	A^5	1869.07	9	0.54	A^5	1888.4	7	0.88	19.33	1.03
lc202	A^5	1788.09	7	0.38	A^5	1967.69	8	1.03	179.6	10.04
lc203	A^4	1746.44	18	0.54	A^5	1973.71	10	1.34	227.27	13.01
lc204	A^4	1858.33	12	0.6	A^4	1828.27	16	1.01	-30.06	-1.62
lc205	A^4	1683.41	11	0.38	A^4	1937.55	12	1.19	254.14	15.10
lc206	A^4	1617.97	9	0.34	A^5	1947.05	16	0.74	329.08	20.34
lc207	A^5	1799.76	8	0.45	A^4	1880.3	14	0.88	80.54	4.48
lc208	A^4	1690.33	14	0.39	A^4	1640.56	9	0.91	-49.77	-2.94
lr101	$A^{23}B^5C$	2634.37	21	3.58	$A^{24}B^3C^2$	2737.37	20	6.38	103	3.91
lr102	$A^{18}B^6$	2276.49	13	1.3	$A^{14}B^{11}$	2391.12	10	4.43	114.63	5.04
lr103	$A^{23}B^2$	2564.69	17	0	$A^{19}B^3D$	2293.27	12	2.84	-271.42	-10.58
lr104	$A^{15}B^2$	1833.79	25	4.1	$A^{17}B^2$	1962.21	25	9.76	128.42	7.00
lr105	$A^{22}BDE^2$	2714.36	39	9.45	$A^{13}B^9DE^2$	2705.69	41	18.01	-8.67	-0.32
lr106	$A^{20}B^6$	2807.63	47	0	$A^{17}B^6$	2275.91	28	14.03	-531.72	-18.94
lr107	$A^{17}B^2$	2020.17	34	5.95	$A^{18}B^2$	2168.01	34	12.86	147.84	7.32
lr108	$A^{15}B^3$	1916.15	26	5.21	$A^{13}B^4$	1938.44	28	8.49	22.29	1.16
lr109	$A^{18}B^3$	2149.12	33	6.64	$A^{17}B^6$	2309.1	38	13.28	159.98	7.44
lr110	$A^{16}B^4$	2104.42	30	6.62	$A^{19}B$	2102.16	33	14.22	-2.26	-0.11
lr111	$A^{13}B^3$	1722.69	9	0.77	$A^{13}B^4$	1819.35	8	1.42	96.66	5.61
lr112	$A^{15}B^2$	1803.25	24	4.24	$A^{15}B^2$	1849.97	25	29.71	46.72	2.59
lr201	A^5	1951.7	16	0.27	A^5	1957.34	13	0.45	5.64	0.29
lr202	A^5	1829.85	10	0.1	A^5	1971.99	15	0.28	142.14	7.77
lr203	A^4	1692.53	14	0.1	A^4	1522.56	10	0.1	-169.97	-10.04
lr204	A^3	1313.93	4	0.06	A^3	1388.62	12	0.18	74.69	5.68
lr205	A^4	1718.77	11	0.2	A^4	1726.76	15	0.26	7.99	0.46
lr206	A^4	1670.16	10	0.19	A^4	1693.89	14	0.27	23.73	1.42
lr207	A^3	1576.28	14	0.03	A^3	1485.79	15	0.04	-90.49	-5.74
lr208	A^3	1324.95	12	0.02	A^3	1314.5	10	0.14	-10.45	-0.79
lr209	A^4	1536.91	12	0.12	A^4	1618.81	10	0.28	81.9	5.33
lr210	A^4	1588.14	10	0.12	A^4	1634.03	16	0.43	45.89	2.89
lr211	A^3	1495.11	8	0.07	A^3	1278.8	7	0.12	-216.31	-14.47
lrc101	A^{26}	2878.3	24	6.35	A^{26}	3039.85	27	15.93	161.55	5.61
lrc102	A^{20}	2413.52	15	1.99	$A^{20}B$	2560.29	15	4.84	146.77	6.08
lrc103	A^{17}	2148.43	15	1.68	$A^{15}D^2$	2229.91	10	2.23	81.48	3.79
lrc104	A^{15}	1867.84	11	1.43	$A^{15}D$	2037.14	16	2.28	169.3	9.06
lrc105	A^{21}	2485.93	19	3.15	A^{25}	2744.14	16	4.27	258.21	10.39
lrc106	A^{19}	2411.83	24	3.68	$A^{20}D$	2563.58	27	8.3	151.75	6.29
lrc107	A^{20}	2633.09	33	0	$A^{19}B^2$	2361.83	23	6.91	-271.26	-10.30
lrc108	A^{17}	2104.33	21	2.74	$A^{17}D$	2171	20	7.97	66.67	3.17
lrc201	A^7	2136.75	17	0.42	A^6	2088.02	20	0.58	-48.73	-2.28
lrc202	A^5	1858.07	11	0.08	A^5	1894.69	7	0.27	36.62	1.97
lrc203	A^5	1506.36	8	0.22	A^5	1632.64	16	0.47	126.28	8.38
lrc204	A^4	1407.35	19	0.29	A^5	1498.75	18	0.45	91.4	6.49
lrc205	A^6	1865.43	15	0.39	A^6	1756.88	22	0.5	-108.55	-5.82
lrc206	A^5	1874.92	11	0.31	A^5	1843.09	19	0.37	-31.83	-1.70
lrc207	A^5	1766.13	16	0.24	A^5	1749.16	18	0.41	-16.97	-0.96
lrc208	A^5	1646.5	13	0.38	A^5	1725.21	25	0.72	78.71	4.78
Average		1675.75	13.76	1.45		1712.28	14.38	3.54	36.53	2.25

ALNS with a full recharge policy implemented.

The situation occurred in which ALNSFR could not find an initial solution due to battery infeasibilities. We therefore increased the battery capacities in steps of 10% until a feasible solution was found. We did this for instances *lr103, lrc103, lrc104, lrc105, lrc106* and *lrc108*. Also, the situation occured in which ALNSFR could not find a conflict-free initial solution. Again, we increased the battery capacities in steps of 10% for the following instances: *lc107, lc108, lr102, lr103, lr104, lr106, lr108, lr111, lr112, lrc101, lrc102, lrc103, lrc104, lrc107* and *lrc202*.

The results on these instances are presented in Table 4. Furthermore, we report the instance (Instance), fleet composition (fleet), objective value ($f(\mathcal{X})$) and the number of CSs visited ($\mathcal{F}$). Also, we report the average number of conflicts during the search ($\mathcal{C}$). Finally, we implement a comparison between the two methods by reporting the difference in objective (Δ_f) and how many percent the ALNSFR is off from the solution found by ALNSPR (% off).

From the table, we observe that the differences between the two policies are not very relevant for the small-sized instances. However, for the large instances, we can argue that ALNSPR performs a bit better than ALNSFR. There are a few instances in which ALNSFR actually outfperforms ALNSPR, but on average ALNSPR has an objective that is about 2.25% lower. This difference is thus less convincing than the one found for the PDPTW-EV: Goeke (2019) found that the full recharge policy was on average 3.84% off from the partial recharge policy. In contrast, it is more convincing than the one found for the EVRPTW: Keskin and Çatay (2016) found that the full recharge policy was on average 1.64% off from the partial recharge policy. However, we should interpret these differences with care as the objective functions of Goeke (2019) and Keskin and Çatay (2016) do not take into account acquisition costs. Furthermore, the number of CSs visited in ALNSFR is on anverage 14.38, compared to 13.76 CSs in ALNSPR. Rounding to an integer number of CSs, there is thus no difference.

Also, we make the interesting observation that the average number of conflicts in ALNSFR was almost 2.5 times as high (3.54 in ALNSFR compared to 1.45 in ALNSPR). One might already have expected the number of conflicts to be higher in ALNSFR because EVs are restricted to recharge fully and cannot recharge a bit less in order to avoid a conflict, while still having enough battery to make it to the next CS.

In general, our results thus confirm the status quo: also for the PDPTW-HEVC, a recharge policy in which any positive amount of energy can be recharged is superior to a policy in which EVs are restricted to always recharge fully.

5.4.3 Conflict analysis

In this section, we restrict our attention to the set of representative instances defined in Section 5.3. The aim is to study the changes in obtained solutions and the number of

Table 5: ALNSPR results on representative instances for different numbers of CSs.

		$\chi = 0$			$\chi = 0.25$			$\chi = 0.50$			$\chi = 0.75$		
Instance	$\|F\|$	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$
c101C6	3	384.44	3	0.24	484.92	2	0	$+\infty$			$+\infty$		
c202C12	5	695.53	4	0.57	697.53	3	0.45	1165.99	3	1.22	$+\infty$		
lc101	21	1950.87	10	1.28	2042.25	12	2.71	2000.74	8	2.23	2660.39	11	1.99
lc201	21	2133.84	10	0.39	2106.42	8	0.57	2242.70	12	0.90	$+\infty$		
lr101	21	2545.82	16	3.05	2561.78	17	3.45	2669.73	20	6.32	2929.88	3	0.59
lr201	21	2202.14	15	0.38	2114.69	7	0.31	2216.72	15	0.62	2374.25	9	0.99
lrc101	21	2896.20	32	6.38	3035.77	28	7.34	3195.27	27	11.69	3300.47	23	12.32
lrc201	21	2196.86	17	0.21	1994.42	16	0.55	2246.40	16	0.66	2339.22	13	1.69
r102C12	4	374.76	2	0	379.38	2	0.11	379.38	2	0.08	$+\infty$		
r201C10	4	434.16	4	0.22	442.75	4	0.35	884.27	4	0.68	$+\infty$		
rc102C12	4	544.58	1	0	550	1	0	550	1	0	$+\infty$		
rc201C10	4	450.58	7	0.08	530.67	6	0.14	960.51	4	0.69	$+\infty$		
Average		1400.82	10.08	1.06	1411.72	8.83	1.33	1682.88	10.18	2.15	2720.84	11.8	3.52
Average*		2358.38	18	2.26	2349.78	16	2.87	2465.77	17.2	4.30	2720.84	11.8	3.52

conflicts as certain aspects of the problem change. To be specific, we consider a decrease in number of CSs and increase in inverse recharge rate and observe the differences in objective value, number of CS visits and the number of conflicts. Both of these changes are expected to increase the number of conflicts and decrease the solution quality. We focus on different values of CSs and different values of g^k because of practical implications: a decision maker might ask itself how optimal routes will differ in an infrastructure with fewer CSs, or while implementing a fleet of EVs that recharge more slowly.

Because we are mainly interested in relative performance, we restrict the run-time to 0.5 minute for the small instances and 2 minutes for the large instances. Again, we decide not to implement a stopping criterion in terms of a maximum number of iterations without improvement.

First we reduce the number of CSs by $\chi 100\%$, for χ values 0.25, 0.50 and 0.75. That is, if the total number of CSs of an instance is $|F|$, then we multiply this number by χ and round to an integer number of CSs that should be removed from the instance. The CSs are removed in a random manner. Our hypotheses are that the number of conflicts will increase and the objective values will deteriorate, if we decrease the number of CSs. In fact, we know that the objective value can never improve once we remove a few CSs, as all solutions with fewer CSs are also allowed under the instance with all the CSs included. If a lower objective is found, this should be due to the fact that the heuristic was more lucky in finding good solutions during the limited run-time.

We run ALNSPR on the set of representative instances for the different values of χ, and report the results in Table 5. If no feasible solution can be found, for any reason, we report the trivial objective value $+\infty$. We note that the reported averages should be interpreted with care, due to the presence of infeasible solutions that are not weighted into the average. Therefore, we also report a corrected average (with asterisk) in case all rows that contain at least one $+\infty$ are omitted.

The results indicate that as χ increases, the objective deteriorates. For the corrected average, as χ increases from 0 to 0.25, the objective first decreases only slightly. Then,

Table 6: ALNSPR results on representative instances for different values of g^k.

		$\omega = 0$			$\omega = 0.25$			$\omega = 0.50$			$\omega = 0.75$		
Instance	g^k	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$	$f(\mathcal{X})$	$\mathcal{F}$	$\mathcal{C}$
c101C6	3.47	384.44	3	0.24	484.44	3	0	544.92	1	0	544.92	1	0
c202C12	3.47	695.53	4	0.57	695.42	4	0.8	712.09	4	0.6	716.59	4	0.66
lc101	3.39	1945.35	11	0.7	1964.99	12	1.47	1890.16	12	1.59	1929.22	10	1.98
lc201	2.28	2121.54	9	0.41	2122.76	7	0.24	2128.3	11	0.41	2108.17	7	0.41
lr101	0.48	2717.36	22	3.89	2712.4	20	3.88	$+\infty$			$+\infty$		
lr201	0.16	2337.3	16	0.48	1994.1	15	0.21	2173.09	10	0.32	2227.74	15	0.44
lrc101	0.38	3011.97	28	4.65	3113.17	30	6.83	3009.64	29	9.24	3167.35	32	9.48
lrc201	0.14	2489.98	19	0.21	2867.07	26	2.93	2721.31	24	3.05	2869.79	32	4.83
r102C12	0.49	374.76	2	0	374.76	2	0	374.76	2	0	374.76	2	0
r201C10	0.49	434.16	4	0.22	434.16	4	0.27	448.21	5	0.25	497.2	5	0.26
rc102C12	0.39	544.58	1	0	544.58	1	0	544.58	1	0	544.58	1	0
rc201C10	0.39	450.58	7	0.08	450.58	7	0.07	458.98	9	0.12	450.58	7	0.14
Average		1458.96	10.5	0.95	1479.87	10.92	1.39	1364.19	9.82	1.42	1402.81	10.55	1.65
Average*		1344.56	9.45	0.69	1367.82	10.09	1.17	1364.19	9.82	1.42	1402.81	10.55	1.65

as χ increases from 0.25 to 0.50, the average objective increases with about 5%. Finally, when χ increases from 0.50 to 0.75, the average objective increases with about 10%. In total, a decrease in the number of CSs by 75% only leads to a total increase in objective of about 15%. These results are only a bit less convincing than the results obtained by Ding, Batta, and Kwon (2015) for the E-VRP, who found an increase in total distance of about 15% for a decrease in the number of CSs by 60%.

Furthermore, we note that there is no clear trend in $\mathcal{F}$. Next to this, we are able to observe a positive trend in $\mathcal{C}$ which seems to flatten out. That is, as χ first increases from 0 to 0.25, the average number of conflicts increases from 2.26 to 2.87. Then it increases to 4.30 (χ becomes 0.5) and then it decreases to 3.52 (χ becomes 0.75).

Now we increase the inverse recharge rate by $\omega 100\%$, for ω values 0.25, 0.50 and 0.75. That is, we decrease the rate by which EVs recharge at a CS. If we look at (22), then we expect the departure times at CSs to increase whenever we increase g^k. Consequently, EVs need to spend more time at every CS before their recharge operation is completed, and our hypotheses are thus that again the number of conflicts will increase and the objective values will deteriorate.

We run ALNSPR on the set of representative instances for the different values of ω, and report the results in Table 6. The results indicate that as ω increases, the objective deteriorates. In particular, as ω increases from 0 to 0.25, the objective increase with about 2%. Then, as ω increases from 0.25 to 0.50, the average objective decreases only slightly. Finally, when ω increases from 0.50 to 0.75, the objective increases with about 3%. In total, for an increase in ω of 75%, the average objective increases by about 4%. These results are less convincing than the results obtained by Ding, Batta, and Kwon (2015), who found an increase in total distance of about 5% already for an increase in g^k by about 50%.

Again, there is no clear trend in $\mathcal{F}$. Also, in line with our hypothesis, the average number of conflicts seems to increase as ω increases. As ω first increases from 0 to 0.25, the average number of conflicts increases from 0.69 to 1.17. Then it increases to 1.42 (ω

becomes 0.50) and then to 1.65 (ω becomes 0.75).

6 Conclusion

In this paper, we introduced the Pickup and Delivery Problem with Time Windows, a Heterogeneous fleet of Electric Vehicles and Capacitated charging stations (PDPTW-HEVC). The problem studies two important aspects which have not previously been considered in the PDPTW. First, it considers a heterogeneous fleet, following the research by Hiermann et al. (2016) for the E-VRP. Due to this extension, we can no longer assume that it is always better to use as few EVs as possible as there are different types of EVs. Second, the problem considers the fact that CSs might be occupied upon arrival. We thus search for solutions in which no conflict occurs at the stations. Consequently, EVs sometimes need to wait at a CS, or even make a detour to another CS.

First, we provided a mathematical formulation of the problem, which a decision maker can implement into a solver. To this end, we discussed important implementation issues. In short, the number of conflict-free constraints explodes as the number of customers and thus recharge paths grows large, such that the exact method runs out of memory. Sometimes the exact method can be aided by giving a priori information on the number of EVs available. For a reasonable set of small instances, we did just that, such that we were able to compare the performance of our proposed ALNS heuristic to the exact method. The ALNS framework itself incorporates a construction heuristic, 7 destroy methods, 3 insertion methods, and a specified heuristic that attempts to resolve every conflict from a set of routes. Also, it incorporates a Simulated Annealing framework.

Comparing the ALNS to the exact method on a subset of the small instances showed that the PDPTW-HEVC is a difficult problem to solve to optimality. While the exact method was able to find optimal results for 18 out of 24 instances, ALNS was able to match these optimal results for only 9 of the instances. On average, ALNS was about 9% off from the exact method in terms of objective value obtained. Also, 7 out of 24 instances could not be solved to optimality within the time limit even when given a priori information on the number of EVs. The results also showed that the strategy of selecting EVs with the lowest acquisition cost when creating a new route, sometimes led to suboptimal results. Sometimes it turned out to be better to use more expensive EVs in optimal routes due to the trade-off between total route distances and total acquisition cost. The strategy of only considering distance during the course of the algorithm and only considering acquisition cost when creating a new route, is therefore not sufficient. The decision maker should, if possible, evaluate all fleet composition possibilities and determine the optimal fleet, similar to what we did in this research for the exact method in Section 5.4.1. Lastly, the comparison demonstrated that PDPTW-HEVC is very sensitive to destroy parameter adjustments. The decision maker should keep this in mind, and modify the destroy parameter values to the instance at hand.

Then we compared a partial- to a full recharge policy and confirmed the notion that

also for the PDPTW-HEVC, the former policy had superior results. However, the latter policy was on average only about 2% off from a partial recharge policy. The results on this comparison are thus less convincing compared to the ones already obtained in PDPTW-EV literature. A full recharge policy barely required more visits to CSs (14.38 compared to 13.76), however the number of conflicts was about 2.5 times as high (3.54 compared to 1.45). A decision maker should thus weigh any benefits of using a full recharge policy to the 2% cost benefit of using the partial recharge policy.

Furthermore, we implemented a conflict analysis in which we studied the changes in obtained solutions as two important aspects of the problem changed: the number of CSs and the rate at which EVs recharge at the CSs. First, as the number of CSs was decreased, the objective increased and the number of conflicts rose. For a decrease in the number of CSs by 75%, the objective only increased by about 15% and the number of conflicts increased from 2.26 to 3.52. These results were similar to results obtained for the E-VRP. The decision maker should thus take into account that when the CS infrastructure is decreased drastically, the heuristic is still able to find solutions that are not very far off from the ones obtained with all the CSs included.

Finally, we decreased the rate at which EVs recharge at the CSs. For an increase in the inverse recharge rate by 75%, the objective only increased by about 4% and the number of conflicts increased from 0.69 to 1.65. These differences in objective were less convincing than the ones found for the E-VRP. The decision maker should thus take into account that for the PDPTW-HEVC, if we introduce a fleet of EVs that recharges about 75% slower, still we are able to find solutions that are not far off from the original solutions.

This research also has its limitations. In Section 5.4.1, we argued that for the small instances, inserting up to two CSs between customers gave the ALNS enough flexibility because we saw that optimal routes consisted of no more than two CSs in between customers. However, one might argue that this amount of flexibility need not be enough for the large instances, for which we cannot observe the optimal routes. Another limitation lies in the fact that we only considered one strategy of always choosing the cheapest available EV when creating a new route. Ideally, we would want to try every possible fleet composition, or at least pick a random EV and run the ALNS many times. As said before, a downside of this is that ALNS needs to run many times. Another limitation, already discussed in Section 4.5, is that we do not consider shifting complete clusters of conflict arrivals forward in time, but we only resolve within-cluster conflicts. Therefore, some solutions that might actually be fine are disregarded, because the shifting heuristic is not flexible enough to make these solutions conflict-free. Finally, our conflict analysis in Section 5.4.3 is limited in the sense that it is only applied to the set of representative instances. Also, we randomly selected CSs to be removed only once. Because some CSs are more important than others, given their locations relative to the customers, a better conflict analysis would be to repeat the process of randomly removing the CSs many times

and study average results.

The introduction of a heterogeneous fleet and capacity on the CSs have made this problem quite complex. At the same time, these aspects are (and could become even more) important in any variant of an electric vehicle routing problem. Our proposed heuristic is not yet able to solve every category of instances of the problem to optimality within reasonable time. The main challenge for realistically sized instances is that in order to find the optimal fleet composition, very many possibilities need to be explored. Future research could focus on this point and aim to find a heuristic that avoids evaluating all fleet possibilities while better taking into account the trade-off between total distance and total acquisition cost. As it is impossible to evaluate all fleet composition possibilities if the number of EVs is reasonably large, this heuristic should incorporate a smart mechanism or local search procedure for finding the optimal fleet. An idea would be to give proportionally higher costs to request insertions on routes that are served by more expensive EVs (and vice versa). Lastly, future research could focus on introducing a variable capacity into the PDPTW-EV, such that each station can allow for multiple simultaneous recharge operations.

References

Andelmin, Juho and Enrico Bartolini (2017). "An exact algorithm for the green vehicle routing problem". In: *Transportation Science* 51.4, pp. 1288–1303.

Belfiore, Patrıcia Prado and Luiz Paulo Lopes Fávero (2007). "Scatter search for the fleet size and mix vehicle routing problem with time windows". In: *Central European Journal of Operations Research* 15.4, pp. 351–368.

Bruglieri, M, S Mancini, and O Pisacane (2019). "The green vehicle routing problem with capacitated alternative fuel stations". In: *Computers & Operations Research* 112.104759.

Conrad, Ryan G and Miguel Andres Figliozzi (2011). "The recharging vehicle routing problem". In: *Proceedings of the 2011 Industrial Engineering Research Conference*. IISE Norcross, GA, p. 8.

Ding, Nan, Rajan Batta, and Changhyun Kwon (2015). *Conflict-free electric vehicle routing problem with capacitated charging stations and partial recharge*. Tech. rep.

Dumas, Yvan, Jacques Desrosiers, and Francois Soumis (1991). "The pickup and delivery problem with time windows". In: *European Journal of Operational Research* 54.1, pp. 7–22.

Erdoğan, Sevgi and Elise Miller-Hooks (2012). "A green vehicle routing problem". In: *Transportation Research Part E: Logistics and Transportation Review* 48.1, pp. 100–114.

Felipe, Ángel, M Teresa Ortuño, Giovanni Righini, and Gregorio Tirado (2014). "A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges". In: *Transportation Research Part E: Logistics and Transportation Review* 71, pp. 111–128.

Froger, Aurélien, Jorge E. Mendoza, Ola Jabali, and Gilbert Laporte (June 2017). *A Matheuristic for the Electric Vehicle Routing Problem with Capacitated Charging Stations*. Research Report. Centre interuniversitaire de recherche sur les reseaux d'entreprise, la logistique et le transport (CIRRELT). URL: `https://hal.archives-ouvertes.fr/hal-01559524`.

Goeke, Dominik (2019). "Granular tabu search for the pickup and delivery problem with time windows and electric vehicles". In: *European Journal of Operational Research* 278.3, pp. 821–836.

Goeke, Dominik and Michael Schneider (2015). "Routing a mixed fleet of electric and conventional vehicles". In: *European Journal of Operational Research* 245.1, pp. 81–99.

Grandinetti, Lucio, Francesca Guerriero, Ferdinando Pezzella, and Ornella Pisacane (2016). "A pick-up and delivery problem with time windows by electric vehicles". In: *International Journal of Productivity and Quality Management* 18.2-3, pp. 403–423.

Hiermann, Gerhard, Jakob Puchinger, Stefan Ropke, and Richard F Hartl (2016). "The electric fleet size and mix vehicle routing problem with time windows and recharging stations". In: *European Journal of Operational Research* 252.3, pp. 995–1018.

Hougardy, Stefan (2010). "The Floyd–Warshall algorithm on graphs with negative cycles". In: *Information Processing Letters* 110.8-9, pp. 279–281.

Keskin, Merve and Bülent Çatay (2016). "Partial recharge strategies for the electric vehicle routing problem with time windows". In: *Transportation Research Part C: Emerging Technologies* 65, pp. 111–127.

Liu, Fuh-Hwa and Sheng-Yuan Shen (1999). "The fleet size and mix vehicle routing problem with time windows". In: *Journal of the Operational Research Society* 50.7, pp. 721–732.

Montoya, Alejandro, Christelle Guéret, Jorge E Mendoza, and Juan G Villegas (2017). "The electric vehicle routing problem with nonlinear charging function". In: *Transportation Research Part B: Methodological* 103, pp. 87–110.

Pisinger, David and Stefan Ropke (2007). "A general heuristic for vehicle routing problems". In: *Computers & Operations Research* 34.8, pp. 2403–2435.

Ropke, Stefan and David Pisinger (2006). "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows". In: *Transportation Science* 40.4, pp. 455–472.

Schneider, Michael, Andreas Stenger, and Dominik Goeke (2014). "The electric vehicle-routing problem with time windows and recharging stations". In: *Transportation Science* 48.4, pp. 500–520.

Solomon, Marius M (1987). "Algorithms for the vehicle routing and scheduling problems with time window constraints". In: *Operations Research* 35.2, pp. 254–265.

Toth, Paolo and Daniele Vigo (2003). "The granular tabu search and its application to the vehicle-routing problem". In: *Informs Journal on Computing* 15.4, pp. 333–346.

Appendix

A Floyd-Warshall algorithm and backtracking procedure

The Floyd-Warshall algorithm computes a shortest path between every pair of vertices in a weighted directed graph (Hougardy (2010)). Let $\mathcal{G} = (V, E)$ be the directed input graph with $|V| = n$. Also, let $c : E(\mathcal{G}) \to \mathbb{R}$ be a function returning the weights of the edges of $\mathcal{G}$. Given that no negative cycles exist in the input graph, the algorithm outputs the correct shortest paths in $\mathcal{O}(n^3)$.

The generic algorithm simply returns the total weight of the shortest path between every pair of vertices. For every pair of vertices i and j, matrix entry $M(i, j)$ contains the shortest path weight. In some applications, we might be interested in the actual path that is travelled, i.e. the vertices that are visited in between the vertex pair belonging to the shortest path. Therefore, we store the intermediate vertices in the matrix B, where entry $B(i, j)$ contains the predecessor of j on a path from i to j.

Algorithm 2 returns the matrices M and B. In the initialization, we set the weights of all shortest paths between distinct vertices equal to infinity. Then, we update the values for these paths by considering all vertices $k \in V$ as an intermediate vertex. Algorithm 3 can be used to print the visited vertices for a vertex pair (i, j) in $\mathcal{G}$. Line 4 contains a recursive call to the algorithm for a path from i to the predecessor of j.

Algorithm 2 Floyd-Warshall algorithm

Input: Graph $\mathcal{G}$
Output: n x n matrix M and n x n matrix B

```
1:  M(i,j) = ∞ ∀i ≠ j
2:  M(i,i) = 0 ∀i
3:  M(i,j) = c(i,j) ∀(i,j) ∈ E(G)
4:  B(i,j) = i ∀(i,j) ∈ E(G)
5:  for k = 1 to n do
6:    for i = 1 to n do
7:      for j = 1 to n do
8:        if M(i,k) + M(k,j) < M(i,j) then
9:          M(i,j) = M(i,k) + M(k,j)
10:         B(i,j) = B(k,j)
11:       end if
12:     end for
13:   end for
14: end for
```

Algorithm 3 Path(i, j)

1: **if** $i = j$ **then**
2: Print: i is visited
3: **else**
4: Do *Path(*i, $B(i, j)$*)*
5: Print: j is visited
6: **end if**

B Construction heuristic for ALNS

The construction heuristic presented below is used at the start of the ALNS algorithm to find a feasible solution to the PDPTW-HEVC.

Algorithm 4 Construction heuristic

Calculate $\mathcal{D}_{i,n+i}$ $\forall i \in P$
Start a new route $\overline{r}$ and add request with pickup i^* having the lowest possible value of $\mathcal{D}_{i,n+i}$ given that $\overline{r}$ must be feasible (if needed, apply *insertCS*)
Set of unhandled pickups for requests $U \leftarrow P \setminus \{i^*\}$
Current route $r \leftarrow \overline{r}$
Initial solution $\mathcal{X} \leftarrow \{r\}$
while U is non-empty **do**
  Calculate Δf_{ir}^{pq} $\forall i \in U$ and for every possible positions p and q, $p < q$
  for all Δf_{ir}^{pq} in increasing order **do**
    if insertion not feasible **then**
      See if we can make it feasible by applying *insertCS*
    end if
    if we can make it feasible **then**
      Insert i and $n+i$ at their optimal position in r
      If needed, insert the CSs
      $U \leftarrow U \setminus \{i\}$
      break for
    end if
  end for
  if no request has been inserted into r **then**
    Start a new route s and add request with pickup $j^* \in U$ having the lowest possible value of $\mathcal{D}_{j,n+j}$ given that s must be feasible
    $U \leftarrow U \setminus \{j^*\}$
    $r \leftarrow s$
    $\mathcal{X} \leftarrow \mathcal{X} \cup \{s\}$
  end if
end while
if *containsConflicts*($\mathcal{X}$) **then**
  resolveConflicts($\mathcal{X}$)
end if
if conflicts cannot be resolved in $\mathcal{X}$ **then**
  No initial solution can be found
end if

C Parameter tuning process

Table 7: Evaluated parameter values for ALNS. Initial parameter value in bold. Optimal parameter value in italic.

Parameter	Values				
w	0.025	**0.05**	*0.075*		
c	0.995	***0.99975***	0.9999		
n_{seg}	50	***100***	500		
σ_1	*23*	**33**	43		
σ_2	*4*	**9**	15		
σ_3	7	**13**	*19*		
r	*0.02*	**0.1**	0.5		
p	0.25	**0.5**	*0.75*		
$\xi_{request}$	*0.2*	0.4	**0.6**	0.8	
ξ_{route}	0.2	***0.3***	0.4	0.6	0.8
ϕ	*3*	**9**	15		
χ	1	***3***	5		
ψ	0.5	**2**	*8*		

D Notes on Java code

Now we present a few notes on the Java code that is used. The used classes are divided into four packages: **Data**, **Main**, **Objects** and **SolutionMethods**.

The package **Data** consists of classes that are needed to read an instance of the PDPTW-HEVC. First of all, the class *HEVCR* contains important info for all objects that define an instance of the problem. In particular, it contains info on the vertices and the EVs. Correspondingly, the classes *Vehicle* and *Vertex* contain information on a specific EV and vertex respectively.

The next package, **Main**, contains the most important classes used. The first class, *GenMatrix*, is used to print a randomly generated matrix of EV characteristics. It is especially useful for large instances and designed for the measure of Hiermann et al. (2016) for scaling the battery capacities based on the EV type. The class *H* is used to construct the set of arcs used in the exact formulation. It constructs an auxiliary graph on the CSs, calculates all shortest paths using Floyd-Warshall and generates the set of arcs. Class *Main* is used as our main class to call the methods. The *Solution* class represents a solution object. Using this class, two different solution objects can easily be compared (e.g. in terms of objective value). The *Toolbox* class incorporates only one method, which calculates the Euclidean distance between any two vertices in two-dimensional space.

Package **Objects** contains all kinds of classes that are used to represent objects of our problem. The *Arc* class defines an arc between any two vertices. The *Arrival* class defines an arrival at a CS. These arrivals are defined by, amongst others, an arrival time and a CS at which this arrival occurs. Also, this arrival can be repositioned (set) to another CS. A conflict is defined by the class *Conflict* and it relates two arrival objects. The *Insertion* class is used mainly to check our insertions into different routes in the ALNS. The *Path* class is used to define a (recharge) path in our graph. Pairs of recharge paths occur in the conflict-free constraints and therefore we make use of the class *PathPair*. *Route* is a class used in the ALNS to define a route object, characterized by the list of vertices visited and the EV that serves this route. *VehiclePair* and *VertexPair* simply define a pair of EVs and vertices respectively.

Finally, the package **SolutionMethods** consists of the classes that are used to solve instances of the PDPTW-HEVC. The classes *ALNSFR* and *ALNSPR* are very similar to each other, apart from the fact that the former incorpates a full recharge policy and the latter a partial recharge policy. That is, the latter has additional method getMinRecharge which calculates the energy needed to get from any CS to the next CS (or depot) on a given route. In contrast, in ALNSFR the amount recharged always equals maximum capacity minus the remaining battery. The last class, *MIP* is used to solve the exact formulation of the problem (a mixed-integer program) using CPLEX.