A Deterministic Annealing Local Search for the Electric Autonomous Dial-a-Ride Problem

Yue Su, Jakob Puchinger, Nicolas Dupin

A Deterministic Annealing Local Search for the Electric Autonomous Dial-a-Ride Problem

Yue Su[a], Jakob Puchinger[a,b,*], Nicolas Dupin[c]

[a] *Laboratoire Genie Industriel, CentraleSupélec, Université Paris-Saclay, France*
[b] *IRT SystemX, Palaiseau, France*
[c] *Laboratoire Interdisciplinaire des Sciences du Numérique, CNRS, Université Paris-Saclay, France*

Abstract

This paper investigates the Electric Autonomous Dial-a-Ride Problem (E-ADARP) which consists of scheduling electric autonomous vehicles (EAVs) to transport users from specific origins to specific destinations within predefined time windows. We propose a Deterministic Annealing (DA) meta-heuristic where efficient local search operators are integrated to enhance the solution's quality. The potential visits to the recharging stations are explicitly handled by a bi-directional insertion algorithm. Computational experiments prove the effectiveness of the proposed algorithm in solving E-ADARP. The experiments are conducted under three scenarios: low, medium, and high energy level restriction, representing the constraint on the minimum level of the battery capacity at the end of the route. For each scenario, adapted instances from the literature are tested and an average gap of 0.58% is achieved compared to the best-known solutions for E-ADARP. Several new best solutions are found on previously solved and unsolved instances. Then, we investigate the effect of allowing multiple visits to the recharging stations. The experiments show that this operation can efficiently decrease the total cost and improve the solution feasibility. Furthermore, we establish new benchmark instances based on literature with up to 8 vehicles and 96 requests, with our algorithm providing feasible solutions that the exact method from the literature cannot solve in a given amount of time. These results are an indicator of the high performance of the proposed algorithm.

Keywords: Dial-a-Ride problem, Electric autonomous vehicles, Deterministic annealing, Local search, Meta-heuristic, Vehicle routing

1. Introduction

With an astounding growth in automobile ownership, a series of transport-related problems appear worldwide. These problems, such as greenhouse gas emissions and urban traffic congestion, have severely impacted both the economy and the environment. For example, the cost of congestion in the United States alone is approximately $121 billion per year, including 5.5 billion hours of time lost and an extra 2.9 billion gallons of fuel burned [1].The increasing emission of greenhouse gas worldwide leads to climate change and the rise of sea levels. To address these concerns, two effective axes have been introduced: using clean energy powered vehicles and providing ride-sharing services. In replacing conventional unsustainably-fueled vehicles with electric powered ones, the rechargeable battery

*Corresponding author. Tel.: +33 686133033
Email addresses: yue.su@centralesupelec.fr (Yue Su), jakob.puchinger@centralesupelec.fr (Jakob Puchinger), nicolas.dupin@lisn.upsaclay.fr (Nicolas Dupin)

as a power source has the advantages of zero greenhouse gas emission, less energy cost per mile, and lower noise [2]. However, limitations, such as driving range and the potential need to recharge on the route, have a direct impact on vehicle routing results when dispatching battery electric vehicles to fulfill pickup and delivery tasks. In this study, we consider the optimal scheduling of the EAVs for user-specific transport requirements.

Ride-sharing services is the second possible axis being investigated. Ride-sharing services can reduce the number of vehicles on the route, rendering transport more efficient, benefiting customers with discounted ride price. Ride-sharing service is therefore a win-win strategy from both economic and environmental aspects. The Dial-a-Ride Problem (DARP) is a class of combinatorial optimization problems where customers can formulate transportation requests from a specified origin (or pickup point) to a destination (or drop-off point). A fleet of vehicles serves customers by providing ride-sharing services, respecting a series of constraints (e.g., time-window constraint, capacity constraint, maximum user ride time constraint). This problem is a variant of the vehicle routing problem and has been widely studied in previous literature (e.g., [3], [4], [5], [6],[7], [8]). However, the studies that combine electric vehicles with DARP are scarce. To the best of our knowledge, the only work explicitly considering the use of EAVs in the context of DARP is the work of [9].

Our study determines the optimal routes in the Electric Autonomous Dial-a-Ride Problem (E-ADARP). A partial recharging policy is employed in our study, and the discharging and recharging rates are assumed to be constant. Additionally, a minimum battery level should be satisfied by vehicles when they return to the depot. All the user requests are known in advance and need to be served.

As E-ADARP extends from classical DARP, the exact methods cannot solve large-scale instances within reasonable computational time. Using soft computing techniques such as heuristics and meta-heuristics is the most viable way to find a near-optimal solution for large-scale instances in an acceptable time frame. We established a meta-heuristic approach, namely Deterministic Annealing (DA) meta-heuristic enhanced by a Local Search (LS) for intensification. The recharging stations are explicitly handled by a bi-directional insertion algorithm. An adaptive update of a threshold value for the acceptation criterion is also applied in each iteration. In numerical studies, we assess the performance of the proposed algorithm on modified Cordeau instances ([10]) that have been supplemented with charging stations and battery specifications. The real open source data provided by Uber Technologies (`https://github.com/dima42/uber-gps-analysis/tree/master/gpsdata`) is also used in the experiments. After verifying the algorithm's effectiveness, we extend the model and consider the multiple visits on the recharging stations. We investigate the effect of allowing multiple visits to the same recharging station in contrast to [9]. Moreover, we introduce the new Ropke instances ([11]) that have been supplemented with problem-related features to conduct large-scale experiments.

The remainder of this paper is organized as follows. The next section presents an overview of related literature. Section 3 provides the problem definition. The meta-heuristic algorithm to solve E-ADARP into near optimum is presented in Section 4. In Section 5, the algorithm performance is first compared to the benchmark provided by Bongiovanni. Next, extensive numerical experiments are conducted on Cordeau instances, Ropke instances and Uber instances and new benchmark results are generated. The paper ends up with the conclusion and future work.

2. Literature Review

There are two major approaches in the literature to solve DARP. One is the exact method approach, and the other is the approximation approach (i.e., heuristic/meta-heuristic method). A detailed literature review on the dial-a-ride problem can be found in [12] and [13]. According to the problem features of E-ADARP, we focus on reviewing the representative works that relate to the following characteristics: heterogeneous vehicle fleets, multiple depots, and the use of electric vehicles.

Parragh [14] formally introduces the heterogeneous DARP (H-DARP) that considers the needs of different users (i.e., staffs, patients) and satisfies their requirements (i.e., normal set, sketchers, wheelchairs) by assigning different types of vehicles. A Variable Neighborhood Search (VNS) is employed and tested on 36 instances with up to four vehicles and 48 requests. Parragh et al. [15] extend the previous work by considering driver-related constraints into H-DARP. The column generation is combined with the VNS method and yields high-quality solutions for realistic instances. Braekers et al. [16] proposed a new variant of DARP by considering multiple depots and heterogeneous vehicle fleets. Each vehicle is assigned to a specific depot, and the vehicle should start from and end up with the same depot. Both the exact method and meta-heuristic method are used in their study, and Deterministic Annealing is first used in solving the DARP. The proposed algorithm is tested on DARP benchmark instances with up to 13 vehicles and 144 requests, and several best-known solutions for unsolved instances are improved. Based on the work of Braekers et al. [16] and Masmoudi et al. [17] develop a hybrid genetic algorithm to solve H-DARP. The algorithm is tested on 92 benchmark instances and 40 newly introduced instances. The average gap of the found solutions with the optimal or best-known solutions is less than 0.5 %.

In the context of electric DARP, Masmoudi et al. [18] consider using electric vehicles in the H-DARP, and the recharging operation is realized by swapping battery from battery-swap stations. They use a realistic energy consumption model to formulate the problem, and three enhanced Evolutionary Variable Neighborhood Search (EVO-VNS) are introduced. Bongiovanni et al. [9] study electric autonomous DARP (E-ADARP), where the authors consider partial recharging policy, detour to recharging stations, and selection of destination depots. In the objective function, they minimize the weighted sum of total travel time and total user excess ride time. They formulate the problem into a three-index, a two-index model, and new valid inequalities tailored by problem-specific constraints are introduced in a branch-and-cut algorithm. Based on standard instances from [10], they establish new instances that are supplemented with recharging stations and related parameters. The exact method proposed in this work can solve instances with up to 5 vehicles and 40 requests into optimality. However, no heuristic or meta-heuristic algorithm has been proposed yet for E-ADARP.

We also review some vehicle routing problems that proposed the heuristic approach to solve VRPs with electric vehicles (e.g., [19], [20], [21], [22, 23]). Among them, Erdoğan and Miller-Hooks [19] first proposed a green vehicle routing problem using alternative fuel vehicles. A set of recharging stations are allowed to be visited during vehicle trips. Two constructive heuristics are designed to obtain feasible solutions and enhanced by means of local search to further reduce the total traveled distance. However, the capacity restrictions and time window constraints are not considered in their model. Based on this work, Schneider, Stenger, and Goeke [20] propose a more complicated model named E-VRPTW. They extend the work of [19] by using electric vehicles and considering limited vehicle capacity

and specified customer time window. A Variable Neighborhood Search (VNS) hybridized by Tabu Search in local search is proposed to address E-VRPTW. The recharging stations are inserted or removed by a specific operator. Similar to [19], the recharged energy is assumed to be linear with the recharging time, and a full recharging policy is applied. All the vehicles in this study are assumed to be identical in terms of vehicle and battery capacity. Goeke and Schneider [21]extend the homogeneous E-VRPTW by considering a mixed fleet of electric and conventional vehicles. A realistic energy consumption model that integrates speed, load, and road gradient is employed. To address the problem, they propose an Adaptive Large Neighborhood Search algorithm using the surrogate function to evaluate violations efficiently. Hiermann, Puchinger, Ropke, and Hartl [22] extend the work of [21] by taking into account the heterogeneous aspect (i.e., fleet composition). They solve the problem by Adaptive Large Neighborhood Search (ALNS). A labeling algorithm is embedded to optimize the positions of recharging stations. The recharging policy considered in this work is also fully recharging with a constant recharging rate. Hiermann et al. [23] extend their previous study by applying a partial recharging policy for a mixed fleet of conventional, plug-in hybrid, and electric vehicles. The engine mode selection for plug-in hybrid vehicles is considered as a decision variable in their study. A layered optimization algorithm is presented. This algorithm combines labeling technics and a greedy route evaluation policy to calculate the amount of energy required to be charged, as well as determine the engine mode and energy types. This algorithm is finally hybridized with a set partitioning problem to generate better solutions from obtained routes.

Based on our review, the use of electric vehicles is widely studied in the VRPs but not yet in the DARPs. Furthermore, there is no meta-heuristic designed to tackle DARP with EAVs. With the future prevalence of electric vehicles and autonomous vehicles, using the meta-heuristic method to solve complex E-ADARP is of great value.

3. Problem Definition

In this section, we recall the E-ADARP definition proposed by [9]. The problem is defined on a complete directed graph $G = (V, A)$ where V presents the set of vertices and A the set of arcs. V is further partitioned into two subsets: $N = 1, 2, \cdots, i, \cdots, 2n$ presents the pickup and drop-off nodes set, and $F = 2n + 1, \cdots, 2n + f$ is the set of recharging stations. It should be noted that the user nodes must be visited exactly once. The depot should be visited at the start and at the end of the tour. If it is necessary, the depot can serve as a recharging station. Moreover, the destination depot is not predefined for each vehicle and can be selected from a set of optional depots. For each user request, a pickup node and a drop-off node is associated. Each request is a couple $(i, n + i)$, where $i = 1, \cdots, n$. All the user requests are known at the beginning of the planning horizon.

For each vehicle $k \in K$, the maximum capacity for each vehicle is denoted as VC_k. For each driver, the maximum route duration is T_{max}. The battery capacity is supposed to be identical for all the electric vehicles, denoted as BC_k. Vehicles can be heterogeneous in terms of vehicle and battery capacity. In our case, we follow the study of [9], and vehicles are assumed to be identical in terms of their vehicle and battery capacity.

For each user node $i \in N$, the time window is defined as $[e_i, l_i]$, in which e_i and l_i represent the earliest and latest time, respectively. Each user pickup and drop-off node is associated with a load q_i and a service duration s_i. As for drop-off point $n + i$, $q_{n+i} = -q_i$ $(i = 1, \cdots, n)$. On the set of origin and destination depot, load and service

duration at the depot is zero. The cost and travel time on each arc is denoted as $c_{i,j}$, and $t_{i,j}$, respectively. The maximum travel duration for request i is denoted as m_i.

At the recharging station, the amount of energy recharged is supposed to be proportional to the time spent at the facilities. The recharging rate of facilities at recharging station f is denoted as α_f, according to their recharging technologies (i.e., fast recharging or slow recharging). As we have mentioned beforehand, the vehicle can be partially recharged while visiting recharging stations and should return to the destination depot with a minimal battery level. We define a minimal battery level ratio γ to present the minimum battery level that vehicles must maintain at the end of routes. The recharging station can only be visited when there is no passenger on-board. Regardless of the features that the electric vehicle fleet inherits, vehicle autonomy should be taken into account. The route duration constraints are not included as we do not need to consider the driver's shift. Moreover, the destination depot will not be predefined at the beginning of the time horizon, and it will be decided during the service.

To summarize, the E-ADARP consists of the following features that are different from the typical DARPs: (1) detour to recharging station on the route, (2) partial recharging at recharging station, (3) vehicle can locate at different origin depots, (4) vehicle can select from a set of optional destination depot, (5) no restriction for route duration time.

The following constraints should be satisfied:

(1) Every route starts from an origin depot and ends at a destination depot;

(2) For each request, its corresponding pickup and drop-off node should belong to the same route, and the pickup node should be visited before its drop-off node;

(3) User nodes and origin depots should only be visited once, while each destination depot can be visited at most once;

(4) The maximum vehicle capacity should be respected at each node;

(5) Each node should be visited within its time window $[e_i, l_i]$ where $i \in V$. Waiting time occurs when the vehicle arrives earlier than the earliest time window;

(6) The maximum user ride time should not be exceeded for any of the users;

(7) The battery level at the destination depot should surpass the minimal battery level;

(8) The energy contains in the vehicle should not exceed the battery capacity;

(9) To consider the user's inconvenience, we set the objective function as a weighted sum of total travel time and total excess user ride time;

For a mathematical formulation of E-ADARP, readers can refer to [9].

4. Meta-heuristic Algorithm

The studied E-ADARP is a combinatorial optimization problem and can hardly be solved exactly for benchmark instances of realistic sizes. Indeed, many aspects (e.g., time window constraints, energy constraints, optimizing visits to recharging stations, recharging time) should be accounted for when evaluating a single route. In order to solve large-scale instances within a reasonable computational time, heuristics and meta-heuristics can be applied. Meta-heuristics have a more global view than heuristics because they allow to deteriorate the current solutions

or even accept the infeasible intermediate solutions. Comparing to heuristics, meta-heuristics are less likely to be trapped in a local optimum and can provide a more profound search.

In this section, we establish a meta-heuristic algorithm, namely Deterministic Annealing (DA) meta-heuristic enhanced by a Local Search (LS) for intensification, to obtain near-optimum solutions for E-ADARP. Deterministic Annealing is a variant of simulated annealing meta-heuristic and is first introduced by [24]. Recent researches show that deterministic Annealing can obtain near-optimal or optimal solutions for a series of vehicle routing problems ([25, 16], [26]). To the best of the authors' knowledge, the only paper that implements deterministic Annealing to solve DARP is that of [16], and it proves the great potential of DA to tackle such problems.

The mechanism of the DA method is illustrated as follows: in each iteration, we apply local search operators to obtain the neighborhood solution x'. For each operator, once the obtained neighborhood solution's cost $f(x')$ is less than the current solution cost $f(x)$, the generated neighborhood solution is accepted. Otherwise, if $f(x') - f(x)$ is smaller than the threshold value T, the generated solution is also accepted. The threshold value is updated in each iteration after the local search and is gradually lowered until zero ([16]). The global best solution is updated by only accepting the solution that is better than the current best solution or the solution that contains more customers (in the case that initial solution cannot insert all customer requests).

There are several reasons for choosing Deterministic Annealing in the meta-heuristic algorithm. The main reason is that: unlike the extensively applied Adaptive Large Neighborhood Search (ALNS), which relies on various parameters (weight factors, operator related parameters) and tedious parameter tuning process, Deterministic Annealing largely relies on a single parameter T, leading to a more efficient application in practice. Secondly, the successful application of Deterministic Annealing in [16] to tackle DARP has proven robust in handling complex DARP. Finally, as mentioned in the literature review, Deterministic Annealing has been applied to tackle neither DARP nor VRP with electric vehicles. It brings us the research potential to explore whether the algorithm can tackle complex E-DARP problem. From our algorithm results, the proposed meta-heuristic performs well compared to the results provided by the exact method.

This section is organized as follows. Firstly, the algorithm structure will be presented, particularly the mechanism of updating the threshold value iteratively and the restart mechanism. Secondly, the constructive heuristic to generate the initial solution will be illustrated. Then, the local search operators will be discussed, where a bi-directional insertion algorithm is embedded in each operator to explicitly determine the insertion of the recharging station. Some techniques used to reduce the neighborhood size is illustrated at the end.

4.1. Algorithm structure

The algorithm structure for the proposed Deterministic Annealing and Local Search (hereafter DA+LS) is presented in Algorithm 1.The algorithm input is the obtained solution from a parallel insertion heuristic. The solution cost is denoted as $c(x)$, and the number of requests served in the solution is Nb_{req}. It should be mentioned that the initial solution found by parallel insertion heuristic is not guaranteed to be feasible; we thus do not consider inserting recharging stations to repair battery infeasibility in this process. The un-inserted requests are called "rejected" requests and will be inserted in the local search process by a specific operator named "*AddNewRequest*". No infeasibility is tolerated during the initial solution construction process. There are basically two steps in the

main algorithm: local search and threshold update. At the beginning of the algorithm, the threshold value T is set to T_{max}, and the best solution x_b and current solution x' are initialized to initial solution x_{init}. The number of iteration is denoted as Nb_{iter}. During the local search process, the local search operators are applied to alter the current solution. In the next step, the threshold value is updated and restart when the value is negative.

In the local search process (line 3 to line 20), the number of operators used depends on the number of assigned requests in the initial solution. In case of existing un-inserted requests, an operator named "*AddNewRequest*" is designed to add these requests into the neighboring solution found by intra-route/inter-route operators (line 15 to line 17). Once all the requests have been properly included in the routes, this operator is deactivated. During the local search process, when encountering battery-infeasible solutions, the bi-directional insertion algorithm is called to insert recharging stations at proper places. Each operator (intra- or inter-route) returns the best neighbor x' from current solution x if it exists. Solution x' is accepted to become the new current solution when the number of assigned requests increases or the total cost is less than that of the current solution plus the threshold value T. Once the new solution x' is accepted, it will also be checked whether it is the best solution x_b or not.

Algorithm 1 Deterministic Annealing Algorithm

Input: Initial solution generated by parallel heuristic

Parameter initialization: $T = T_{max}$, n_{imp}, Nb_{iter}, $i_{imp} = 0$,$iter = 0$;

$x = x_b = x_{init}$;

$c(x_b) = c(x) = c(x_{init})$;

$Nb_{req} = \sum_{k \in K} \sum_{i \in P^u} v_{i,k}$;

Output: Improved feasible solution x_b;

1: **while** $iter \leqslant Nb_{iter}$ **do**
2: $i_{imp} \leftarrow i_{imp} + 1$;
3: **for** $j = 1 \rightarrow n_{oper}$ **do**
4: Apply local search operator on x to obtain neighboring solution x';
5: **if** x is battery-infeasible solution **then**
6: bi-directional insertion algorithm to repair in-feasibility;
7: **end if**
8: **if** $c(x') < c(x) + T$ **then**
9: $x \leftarrow x'$;
10: **end if**
11: **end for**
12: **if** $Nb_{req} \leqslant Nb_{user}$ **then**
13: $n_{oper} \leftarrow AddNewRequest$
14: **end if**
15: $iter \leftarrow iter + 1$
16: $Nb'_{req} \leftarrow \sum_{k \in K} \sum_{i \in P^u} v'_{i,k}$
17: **if** $c(x') < c(x_b)$ **and** $Nb'_{req} = Nb_{req}$ **or** $Nb'_{req} \geqslant Nb_{req}$ **then**
18: $x_b \leftarrow x'$

19: $i_{imp} \leftarrow 0$
20: **end if**
21: **if** $i_{imp} > 0$ **then**
22: $T \leftarrow T - T_{max}/T_{red}$
23: **if** $T < 0$ **then**
24: $r \leftarrow$ random number between 0 and 1
25: $T \leftarrow r \times T_{max}$
26: **if** $i_{imp} > n_{imp}$ **then**
27: $x \leftarrow x_b$
28: $i_{imp} \leftarrow 0$
29: **end if**
30: **end if**
31: **end if**
32: **end while**
33: **return** x_b

In the threshold update process (line 21 to line 31), when no global best solution is found, the threshold value is reduced by T_{max}/T_{red}, where T_{red} is a predefined parameter. When T becomes negative, the threshold value is reset to $r \times T_{max}$, with r a random number generated between zero and one. The search is restarted from x_b when T is negative, and no improvement is found in n_{imp} iterations.

4.2. Insertion Heuristic

We use a parallel insertion heuristic to obtain the initial solution. In the first step, we randomly generated m routes ($0 < m < K$ with K being the number of total vehicles). All the requests are sorted in increasing order with regards to their earliest time window e_i. Each of the m first requests in the sorted list are assigned randomly to different routes. These requests are deleted from the original list. Next, we try to insert the rest users one by one into the routes with regards to various constraints.

The insertion process is based on two steps. In the first step, the distance between the last assigned element in each existing routes with the first element in the list is calculated and sorted in increasing order. Then, route by route, the new users will be examined its feasibility. If this user can be inserted into one route, its pickup and drop-off nodes will be inserted at their best positions (i.e., the position that has minimum increase on total cost) in this route. We delete this customer node in the sorted list and move to the next. If the insertion of the customer node is infeasible, then this customer node will be kept in the list, and we move to the next. If some users are still not assigned, a new route is activated, and the above process will be repeated. The algorithm below explicitly describes the whole insertion process.

4.3. Local Search Operators

Seven operators have been applied to improve the initial solution generated from the constructive heuristic. Among them, three are intra-route operators (i.e., *exchange-pickup*, *exchange-drop-off*, *exchange-two-neighboring-node*), three are inter-route operators (i.e., *relocate*, *exchange*, and *2-opt*). We don't apply the complex intra-route

Algorithm 2 Parallel Insertion Heuristic

Input: Initialize the set of non-assigned users sorted in increasing order of earliest pickup time;

Randomly selected m vehicles and assign the first m users to different vehicles;

Output: x^*: Optimal routing schedule for each vehicle

```
repeat
    Select the first non-assigned user from the list;
    Sort vehicles in increasing order of distance between the last assigned user and the selected user;
    Select the first vehicle in the list;
    repeat
        Check the feasibility of the user's pickup and drop-off nodes;
        if one or more feasible options are found then
            Insert the user in the best position;
        else
            Select the next vehicle in the list;
        end if
    until the user is inserted;
    Remove the user from the list of non-assigned users;
until any user cannot be inserted into any routes;
```

operator such as ejected chains (e.g., [27]) as the advanced operator can actually be realized by elementary moves, which is more computationally efficient in the case of large iteration times. To tackle the requests that could not be inserted into any routes during the constructive heuristic, the *AddNewRequest* is applied in each iteration to try to add un-inserted requests into routes.

Exchange pickup aims at swapping the position of two consecutive points (i, j), which point i is a pickup node and point j is not the corresponding drop-off node. In each iteration, one pickup node is selected randomly. If the succeeder of this pickup node does not corresponds to its drop-off node, then the two positions are exchanged. The following Figure is an example of applying this operator.

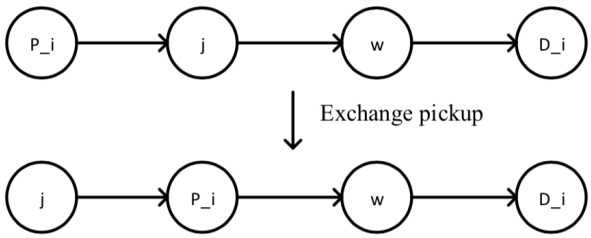

Figure 1 Exchange pickup

Exchange drop-off is designed to swapping the position of two neighboring nodes (i, j), in which point j is a drop-off node and point i is not the corresponding pickup node. In each iteration, one drop-off node is selected randomly, if the precedent node of this drop-off node does not correspond to its pickup node, then the two positions

are exchanged.

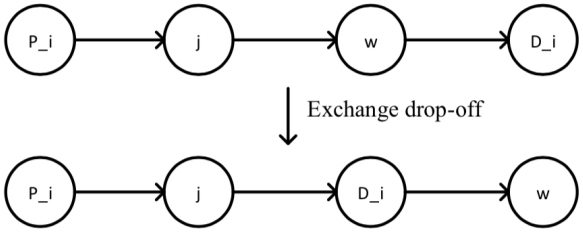

Figure 2 Exchange dropoff

There is another situation shown in the following Figure, where the successive node of pickup node P_i is its drop-off D_i, and the previous node of drop-off node D_j is its corresponding pickup P_j, but we can still exchange D_i and P_j to obtain a new solution. This operation is realized by *exchange-two-neighboring-node*.

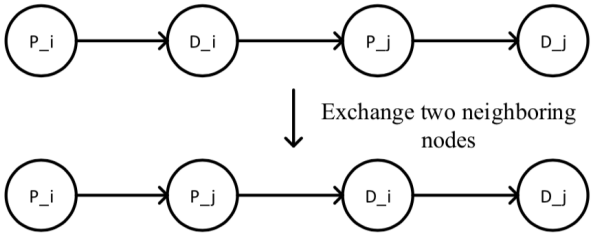

Figure 3 Exchange two neighbor

Relocate operator removes a user request from its current route and re-insert the request in the best position in the route of another vehicle. The operator is applied on a randomly selected user request of a single randomly selected route. It should be mentioned that this operator may eliminate a route if the selected route contains only one request. When the number of routes activated in the solution is less than the number of available vehicles, assigning a request to an idle vehicle is acceptable.

The *exchange* operator swaps two requests of two different routes. The pickup (drop-off) vertex of the first route can only be inserted in the same position as the pickup (drop-off) vertex of the second route. The re-insertion of the second request to the first route should be fulfilled at the best insertion position.

The 2-opt operator selects two random routes (denoted as $route_1$,$route_2$) and removes an arc from each of them. The removed arc is connected with the remaining part of the other route in the route pair. That is, the first part of the $route_1$ is connected with the second part of the $route_2$; the first part of the $route_2$ is connected with the second part of the $route_1$.

AddNewRequest operator is applied in each iteration, trying to insert the "rejected" requests into routes. While there are no more "rejected" requests to be inserted, this operator is deactivated.

4.4. Bi-directional Insertion Algorithm

Each operator is embedded with the bi-directional insertion algorithm when there is a battery-infeasible solution to be repaired. The algorithm structure is shown in the following pseudo-code. There are two algorithms included

in the bi-directional insertion algorithm, namely the forward insertion algorithm (shown in 4) and the backward insertion algorithm (shown in 5). Their corresponding pseudo-codes are depicted below.

In the bi-directional algorithm (3), line 1 to line 7 determines the position list that can be inserted in a recharging station. As we stated in the previous section, the vehicle can only go to the recharging station when the vehicle load is zero. Since the orders of nodes on the route is fixed, the possible positions of recharging stations are also fixed. We first calculate the energy list that contains the energy levels for nodes. If all the energy levels are positive, we directly use the backward insertion algorithm to insert recharging stations from the end of route. Otherwise, the forward insertion algorithm is then called to repair the battery negativity. Once the energy levels are repaired to be positive, the backward insertion algorithm is employed to generate a battery feasible solution x_r.

Algorithm 3 Bi-directional Insertion Algorithm Structure

Input: Battery infeasible solution x;

Output: Repaired feasible solution x_r;

1: $cap = 0$,$pos = \emptyset$;
2: **for** $i = 1 \rightarrow$length(route) **do**
3: $cap = cap + q_{route[i]}$;
4: **if** $cap = 0$ **then**
5: $pos = pos \cup i$;
6: **end if**
7: **end for**
8: Calculate energy list that contains energy level at each node;
9: **if** energy levels in the route are positive **then**
10: Backward Insertion Algorithm to repair the minimum battery level constraint;
11: **else**
12: Forward insertion algorithm to repair the negative battery level;
13: Adjust the values of elements in pos;
14: Backward Insertion Algorithm to repair the minimum battery level constraint;
15: **end if**
16: **return** x_r

The forward insertion algorithm (4) is applied to repair the battery-positive constraint. It starts from the last zero-load node that has a positive energy level. The recharging time (denoted as RT) is calculated as the minimum of time required to recharge to satisfy the battery constraint (denoted as $RequiedTime$) and time allowed by respecting the time window on the next visited node (denoted as $AllowedTime$). If RT is not zero and there is at least one reachable recharging station from this position, then the recharging station that has the minimum detour is inserted. Otherwise, the solution cannot be fixed and the insertion process is terminated. Each time when inserting a recharging station, the energy list is updated. Once the energy level at the end depot is positive, the last inserted recharging station position will be recorded and the forward insertion is ended.

We use the backward insertion algorithm (5) to repair the minimum battery level infeasibility. Firstly, We reverse

Algorithm 4 Forward Insertion Algorithm

Input: Energy negative solution x;

Output: Energy-positive route x', last inserted position index $index_{end}$, updated energy list;

Calculate last energy-positive node index;
while end depot energy is negative **do**
    $RT = max\{0, min\{RequiredTime, AllowedTime\}\}$;
    if RT is not zero **and** there exists reachable recharging stations **then**
        Inserting the recharging station that has minimum detour;
        if energy constraint satisfied **then**
            Record $index_{end}$;
            Update energy list;
        else
            Recharge RT;
            Update energy list;
            Move to the next position;
        end if
    else
        Solution cannot be fixed, return $\emptyset, \emptyset, \emptyset$;
    end if
end while
return x'**,** $index_{end}$, updated energy list;

the position list and start to insert a recharging station from the end of the route (line 2). For each position, the recharging time RT is calculated similar to the forward insertion algorithm. If RT is not zero and there is at least one reachable recharging station from this position, then the recharging station that has the minimum detour is inserted. Otherwise, the solution is discarded. After inserting the recharging station, the energy list is updated and battery constraints are re-examined. If the energy constraint is satisfied, this solution is returned. Otherwise, the vehicle can only recharge RT, and the next possible position is considered. The algorithm is ended when the time window constraint is violated ($RT = 0$) or all the possible positions have been tested. If the solution cannot be repaired after trying all possible positions, the solution is discarded.

Algorithm 5 Backward Insertion Algorithm

Input: Possible position set pos and $index_{end}$, energy-positive route x', updated energy list;

Output: Repaired feasible solution x_r;

```
1: violation = true;
2: Reverse pos;
3: for i = 1 → index_end do
4:     RT = max{0, min{RequiredTime, AllowedTime}};
5:     if RT is not zero and there exists reachable recharging stations then
6:         Inserting the recharging station that has minimum detour;
7:         if energy constraint satisfied then
8:             violation = false
9:             Return repaired solution, violation;
10:        else
11:            Recharge RT;
12:            Update energy list;
13:            Move to the next position;
14:        end if
15:    else
16:        Discard solution;
17:    end if
18: end for
19: if violation == true then
20:    Discard solution;
21: end if
```

4.5. Reducing Neighborhood Size

In the local search process, the neighborhood is searched by using different operators. As we iterate the meta-heuristic thousands of times to find high-quality solutions, the size of the neighborhood has a direct impact on computational efficiency. In order to decrease the computation time, several technics are applied to reduce the size of the neighborhood, including time window tightening, arc elimination ([10]), and customer correlations measure ([28])

A time-window tightening process is executed: (1) for user pickup nodes, e_i was set to $max\{e_i, e_{n+i} - m_i - s_i\}$ and $l_i = min\{l_{n+i} - t_{i,n+i} - s_i, l_i\}$; for user drop-off nodes, $e_(n+i) = \max\{e_{n+i}, e_i + t_{i,n+i} + s_i\}$, and $l_{n+i} = \min\{l_i + m_i + s_i, l_{n+i}\}$, (2) for recharging station, the time window can be tightened by considering the travel time from the origin depot to recharging station and from recharging station to the destination depot. The earliest time to start service at charging station f is set to $\min\{e_j + t_{j,f}\}$, where j is the origin depot; the latest time at charging station f to start service at recharging station is $\max\{T_p - t_{f,j}\}$, where j is the destination depot, (3) for origin and destination depot, the earliest time window e_i is set to $\max\{0, \min\{e_j - t_{i,j}\}\}$, where j is user pickup node set, and $l_i = \min\{l_i, \max\{l_j + s_i + t_{j,i}\}\}$, where j is user drop-off node set.

The arc elimination process follows the method of [10]. Several arcs can be removed from the graph as they will not lead to a feasible solution. This operation significantly decreases the neighborhood size and computational time.

Finally, to further improve computational efficiency, we make reference to [28] and apply a customer-correlation measure to reduce the number of neighbors considered. A set of "promising" arcs are calculated for each customer node using the following formula:

$$r(u,v) = d_{u,v} + r^{WT} \times \max\{e_v - d_u - t_{u,v} - l_u, 0\} + r^{TW} \times \max\{e_u + d_u + t_{u,v} - l_v, 0\}$$

Where r^{WT} is the parameter associated with waiting time, r^{TW} is the parameter associated with time window violation. The set size of promising arcs is set to 40 and $r^{WT} = 0.2$,$r^{TW} = 1.0$ as used in [28].

5. Numerical Experiments

In this section, we conduct extensive numerical experiments and analyze the results. All the algorithms are implemented in Julia 1.3.0 and performed on a 3.20 GHz Intel Core computer with 32G RAM. Three sets of instances, namely adapted instances from [10], [9], and [11] are considered (hereafter called adapted Cordeau, Uber, and Ropke instances), and three different scenarios (i.e., low-energy, medium energy, and high-energy restriction) for each set of instances are discussed. Algorithm results are compared to the exact results presented in [9]. This section is organized as follows. The benchmark instances used to test the algorithm performance are introduced in the first part. Then the sensitivity analysis is conducted on the existing instances to find good parameter settings for the parameters in deterministic Annealing. Additionally, we illustrate the design decisions on the main components of our algorithm by analyzing the contribution of each operator on solution quality. After ensuring the robustness of parameters and operators, we first compare separately the results of the algorithm with that of the exact model under different scenarios on each set of instances. To keep the results comparable, we restrict visits on recharging stations as the authors did in [9]. Then, we relax the constraint on visits to the recharging stations and we compare the results with these in the previous part. The effect of allowing multiple visits on the recharging station is analyzed. Based on previous literature [11], we establish adapted new instances with problem-related characteristics. Finally, the proposed algorithm is performed on the newly-introduced instances to provide feasible solutions under different energy restrictions.

5.1. Benchmark Instances

Three sets of instances are considered in the experiments: (1) the standard DARP benchmark instance set (instance name starts with "a") from [10] supplemented with features of electric vehicle and recharging stations. (2) instances based on the ride-sharing data from Uber Technologies (instance name starts with "u") that adopted from [9], and (3) the adopted large DARP benchmark instances from [11] are also included (with up to 8 vehicles and 96 requests). In each set of instances, three different minimum energy restrictions, namely $\gamma = 0.1, \gamma = 0.4, \gamma = 0.7$, representing respectively low-energy, medium-energy, and high-energy restriction scenario are discussed. We present the algorithm results under each of the energy settings. The higher the value of γ, the stricter the requirement for the energy maintained when vehicles return to the depot.

The instances adopted from [10] are supplemented with recharging station ID, vehicle capacity, battery capacity, the final state of charge requirement, recharging rates, and discharging rates. The same operation is applied to instances adopted from [11] to generate large-scale set of instances. The vehicle capacity is set to three passengers, and the maximum user ride time is 30 minutes. Recharging rates and discharging rates are all set to 0.055KWh per minute according to the design parameter of EAVs provided in: https://www.hevs.ch/media/document/1/fiche-technique-navettes-autonomes.pdf. The efficient battery capacity is set to 14.85 KWh, and the vehicle can approximately visit 20 nodes without recharging.

The ride-sharing dataset of Uber is obtained from the link: https://github.com/dima42/uber-gps-analysis/tree/master/gpsdata. The Uber instances are created by extracting origin/destination locations from GPS logs in the city of San Francisco (CA, USA) and applying Dijkstra's shortest path algorithm to calculate the travel time matrix with a constant speed setting (i.e., 35km/h). Recharging station positions can be obtained through Alternative Fueling Station Locator from Alternative Fuels Data Center (AFDC). For a more detailed description of instances development, the interested reader can refer to [9]. The preprocessed data that extract requests information from the raw data provided by Uber Technologies are published on the EPFL website (https://luts.epfl.ch/wpcontent/uploads/2019/03/e_ADARP_archive.zip).

5.2. Parameter Adjusting

The performance of the proposed algorithm depends on several parameters that should be set in advance. We should therefore identify the robust parameter settings in order to ensure the algorithm performance.

The related parameters are (1) Number of iterations Nb_{iter}, (2) Maximum threshold value T_{max}, (3) Threshold reduction value T_{red}, (4) Restart parameter n_{imp}.

To avoid re-tuning T_{max} when using different types of instances, we use a relative value for T_{max}. The maximum threshold value is expressed as the product of the average distance between two nodes in the studied graph (denoted $\bar{c}$) and a predefined parameter t_{max}, that is $T_{max} = \bar{c} \times t_{max}$, where t_{max} is set to 1.5. Other initial parameter values are set to $T_{red} = 300$ and $n_{imp} = 50$ based on the preliminary results. Firstly, the number of iterations Nb_{iter} is investigated under these parameter settings, and the results are shown in Table 1. To identify the iteration times, it is enough that we select one energy-level restriction (i.e., $\gamma = 0.1$) and analyze the results. We test six values of Nb_{iter}, and the average gap between the best solution results over five runs on 14 instances that the number of requests ranging from small to large are reported. The algorithm results are compared with the best results so

far. Based on the algorithm results, $Nb_{iter} = 5000$ seems to strike a good balance between solution quality and computational time and we select this value to conduct the following experiments.

Table 1

Sensitivity analysis for the number of iterations

Nb_{iter}	1000	2000	3000	4000	5000	6000
Average gap (%)	1.31%	0.93%	0.65%	0.49%	0.40%	0.38%
Average CPU (s)	46.47	100.83	136.98	179.28	203.43	247.69

The values of t_{max}, T_{red} and n_{imp} are correlated, which results in a huge number of combinations. Among these parameters, t_{max} mainly impacts the solution quality ([16]). Therefore, in this part, we concentrate on investigating the best value of t_{max}. We set seven values for t_{max} and for each value of t_{max}, we perform five runs on each instance. To have a global view for the parameter performance, we also run the instances while considering different levels of energy restrictions ($\gamma = 0.1, \gamma = 0.4, \gamma = 0.7$). The corresponding results are summarized in Table 2. For each value, the solution gap between the best solutions of the proposed algorithm over five runs and the exact results is calculated.

Table 2

Sensitivity analysis for t_{max}

t_{max}	0.6	0.9	1.2	1.5	1.8	2.1	2.4
Average gap (%)	1.97 %	1.45%	1.2%	0.51%	1.16%	2.07%	2.11 %
Average CPU (s)	227.51	228.52	234.11	231.04	238.58	258.94	269.33

From the above Table, $t_{max} = 1.5$ can provide us with good solution quality and acceptable computational time. For values of T_{red} and n_{imp}, we keep the initial settings.

5.3. Meta-heuristic Algorithm Design Decision

As the algorithm relies largely on local search operators, the usefulness of operators should be verified. In this part, for each operator, we compare the objective function values over five runs when excluding this operator. All E-ADARP instances (Cordeau instances, Uber instances) are tested, and the solution gap from the exact results are calculated. Results for all inter- or intra-route operators are summarized in Table 3, with average solution gaps and computational time being reported. We can find that each operator performs well in improving the solution quality, especially the 2-opt operator. Therefore, it is necessary to include these operators in local search. As for *AddNewRequest*, it is essential for inserting the un-inserted requests of constructive heuristic. From the above analysis, the usefulness of each local search operator is proved.

Table 3

Contribution of operators

Operator	Average gap (%)	Average CPU (s)
base	0.37%	203.43
2-opt	4.14%	217.03
exchange	0.75%	214.45
relocate	1.45%	155.38
exchange pickup	0.64%	190.59
exchange drop-off	0.68%	207.95
exchange neighboring	0.62%	201.70

5.4. Algorithm performance on E-ADARP Instances

Our proposed algorithm is tested on the existing E-ADARP instances including adopted instances from Cordeau's work ([10]) and Bongiovanni et al. [9]. To verify the robustness of the algorithm, under each energy restriction setting, we compare the results of the algorithm on each instance to the results of three-index and two-index model in [9]. On each instance, we perform ten runs and report on the best obtained objective values and average CPU time. In order to keep the results comparable to the reported results, in this section, we only allow each recharging station to be visited at most once. For each Table in this section, "BC" means the best solution cost, "AT" presents the average computation time, "Obj" is the objective function value of solutions obtained by the three-index or two-index exact model.

5.4.1. Low-energy restriction scenario: γ =0.1

Table 4 shows the results of the proposed Deterministic Annealing and Local Search (hereafter DA + LS) algorithm and the three-index and two-index exact model. We run the algorithm on adapted Cordeau and Uber instances while setting a low-energy restriction for vehicles returning on the depots. For the sake of clarity, we report the average gaps and CPU time between proposed DA + LS results and three-index/two-index exact model results. Detailed results on each instance are shown in Appendix 7.1

Table 4

Results comparison on Cordeau and Uber instances $\gamma = 0.1$

Cordeau instances	DA + LS algorithm	Three-index model[9]	Two-index model[9]
AT(s)	203.43	4505.23	1210.54
Gap	-	NA[a]	0.37%
Uber instances	**DA + LS algorithm**	**Three-index model[9]**	**Two-index model[9]**
AT(s)	281.71	5422.16	1280.83
Gap	-	NA[a]	0.69%

[a] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

From the results, the average gaps between the proposed algorithm and the exact two-index model are quite small for both of the instance sets (i.e., 0.37 % and 0.69 %). For Cordeau instances, the proposed algorithm can find optimal solutions for 9 out of 14 instances, while the exact three-index model can only solve 6 instances optimally and 3 instances cannot be solved within a two-hour time limit. Thus, it is not fair to calculate the solution gap between our algorithm results with the three-index model as the number of solved instances is not equal and we fill the table with "NA".

5.4.2. Medium-energy restriction scenario: $\gamma = 0.4$

The average gaps between proposed DA + LS and three-index/two-index exact model results when $\gamma = 0.4$ are presented below. Detailed results on each instance can be found in Appendix 7.2

Table 5

Results comparison on Cordeau and Uber instances $\gamma = 0.4$

Cordeau instances	DA + LS algorithm	Three-index model[9]	Two-index model[9]
AT(s)	265.58	4547.79	1233.47
Gap	-	NA[a]	0.64%
Uber instances	DA + LS algorithm	Three-index model[9]	Two-index model[9]
AT(s)	324.84	5451.17	2169.25
Gap	-	NA[a]	0.57%

[a] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

From the above results, when we strict the energy requirement for vehicles returning to the depots, the average computational time increases, indicating the increasing complexity to find the feasible solutions. Compared to the results of the exact two-index model, the average gap on Cordeau instances is 0.64%, and 0.57 % on Uber instances. All the instances can be solved with our algorithm and 6 out of 14 are optimally solved. Additionally, we have found a better solution (i.e., a3-24) comparing to the two-index model result.

5.4.3. High-energy restriction scenario: $\gamma = 0.7$

Table 6 shows the results of the DA + LS algorithm and the three-index and two-index exact model on Cordeau instances under high-level energy constraint. The columns named "3index" and "2index" represent the gaps between the heuristic results and the three-index model or two-index model results, respectively. The bold numbers indicate the optimal solutions. Surprisingly, we find three new best solutions (marked in bold with a star) compared to the the two-index model solutions. Furthermore, we observe that some of the instances cannot be solved even though we run the algorithm ten times. The explanation is: in this scenario, vehicles should keep 70% energy while ending their trips, therefore necessitating the visits to the recharging stations in the middle of trips. However, we restrict visits to one per recharging station, cutting off the neighborhood solutions of multiple visiting recharging stations. We have shown in the later section that relaxing the visit to the recharging station can efficiently improve the feasibility.

Table 6

Results comparison on Cordeau instances $\gamma = 0.7$

Instance	DA + LS algorithm		Three-index model[9][a]		Two-index model[9][a]		Gap%	
$\gamma = 0.7$	BC(min)	AT(s)	Obj(min)	CPU(s)	Obj(min)	CPU(s)	3index	2index
a2-16	**240.66**	172.46	240.66*	29.4	240.66*	5.4	0	0
a2-20	NA	427.00	NA	7200	NA	7200	NA	NA
a2-24	364.19	337.16	358.21*	3539.4	358.21*	961.2	1.67 %	1.67 %
a3-18	**240.58**	111.07	240.58*	642.6	240.58*	48	0	0
a3-24	281.82	237.78	277.72*	2957.4	277.72*	152.4	1.48%	1.48%
a3-30	NA	232.04	NA	7200	NA	7200	NA	NA
a3-36	NA	321.11	NA	7200	494.04	7200	NA	NA
a4-16	**223.13**	51.17	223.13*	2179.2	223.13*	67.2	0	0
a4-24	318.29	122.56	321.03	7200	318.21*	1834.8	-0.85%	0.03%
a4-32	**427.92***	206.37	NA	7200	430.07	7200	NA	-0.50%
a4-40	NA	310.56	NA	7200	NA	7200	NA	NA
a4-48	NA	371.45	NA	7200	NA	7200	NA	NA
a5-40	**434.49***	236.87	NA	7200	447.63	7200	NA	-2.94%
a5-50	**625.95***	328.42	NA	7200	NA	7200	NA	NA
Avg		224.11		5296.29		4333.5	NA[b]	NA[b]

[a] Results on a 3.6 GHz Intel(R) Core(TM) with 16 Gb of RAM.

[b] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

* Numbers in bold with star indicate new solution found by proposed algorithm.

We also compare the algorithm results with the exact-model results on 14 Uber instances under high-energy restriction. The results are summarized in Table 7.

Table 7

Results comparison on Uber instances $\gamma = 0.7$

Instance	DA + LS algorithm		Three-index model[9][a]		Two-index model[9][a]		Gap%	
$\gamma = 0.7$	BC(min)	AT(s)	Obj(min)	CPU(s)	Obj(min)	CPU(s)	3index	2index
u2-16	**59.19**	456.79	59.19*	1545.6	59.19*	338.4	0	0
u2-20	**56.86**	701.75	56.86*	133.8	56.86*	72	0	0
u2-24	NA	335.67	NA	7200	NA	7200	NA	NA
u3-18	**50.99**	121.59	50.99*	481.2	50.99*	24	0	0
u3-24	68.98	304.85	69.30	7200	68.39*	400.2	-0.46%	0.86%
u3-30	78.83	571.51	80.35	7200	78.14*	3401.4	-1.89%	0.88%
u3-36	107.13	518.45	NA	7200	105.79	7200	NA	1.27%
u4-16	**53.87**	69.68	53.87	7200	53.87*	88.8	0	0
u4-24	92.11	130.47	89.96*	6045.6	89.96*	22.8	2.39%	2.39%
u4-32	100.53	162.48	NA	7200	99.50*	2827.2	NA	1.04%
u4-40	NA	257.65	NA	7200	NA	7200	NA	NA
u4-48	NA	429.58	NA	7200	NA	7200	NA	NA
u5-40	NA	239.94	NA	7200	NA	7200	NA	NA
u5-50	144.75	276.42	NA	7200	144.36	7200	NA	0.27%
Avg		326.92		5729.01		3598.2	NA[b]	0.67%

[a] Results on a 3.6 GHz Intel(R) Core(TM) with 16 Gb of RAM.

[b] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

Similar results can be found in the above Table. The average gaps between the results of the proposed algorithm with the results of exact two-index model is 0.67 %. Our algorithm shows its competence when tackling instances under higher energy restriction and larger-sized instances.

5.5. Effect of Allowing Multiple Visits to a Recharging Station

In the previous part, we analyze the algorithm results when allowing one visit per recharging station. In the real world, it is more practical that one recharging station is visited multiple times by a vehicle. In this part, we relax the constraint for visiting the recharging stations. Similarly, three scenarios, namely low-energy, medium-energy, and high-energy restriction are taken into account. In each scenario, we perform 10 runs on each instance and compare the results with the best results of the heuristic algorithm in the previous part (restrict the visit to the recharging station).

5.5.1. Low-energy restriction scenario: $\gamma = 0.1$

The experiment results under low-energy restriction are summarized in the following Table.

Table 8

Results comparison: multiple visits v.s. single visit under $\gamma = 0.1$

$\gamma = 0.1$	Multiple visits	Single visit	Gap%	$\gamma = 0.1$	Multiple visits	Single visit	Gap%
Instances	BC(min)	BC(min)	Obj	Instances	BC(min)	BC(min)	Obj
a2-16	237.38	237.38	0	u2-16	57.61	57.61	0
a2-20	279.08	279.08	0	u2-20	55.59	55.59	0
a2-24	346.21	346.21	0	u2-24	91.27	91.27	0
a3-18	236.82	236.82	0	u3-18	50.74	50.74	0
a3-24	**274.80****	274.81	-0.004%	u3-24	**67.87****	68.05	-0.26%
a3-30	413.27	413.27	0	u3-30	77.09	76.75	0.44%
a3-36	481.40	481.40	0	u3-36	**104.21****	105.50	-1.22%
a4-16	222.49	222.49	0	u4-16	53.58	53.58	0
a4-24	310.84	310.84	0	u4-24	90.85	90.72	0.14%
a4-32	**394.81****	395.36	-0.14%	u4-32	**99.58****	99.77	-0.19%
a4-40	456.53	453.84	0.59%	u4-40	134.94	134.17	0.57%
a4-48	**562.78****	563.45	-0.12%	u4-48	**149.45****	149.73	-0.19%
a5-40	418.23	418.23	0	u5-40	**123.23****	123.57	-0.28%
a5-50	**570.09****	571.49	-0.24%	u5-50	**145.85****	146.68	-0.57%
Avg			0.006%				-0.11%

** Double stars indicate by allowing multiple visits, we find lower-cost solution or solve new instances.

With the low-energy restriction, all the instances can be solved optimally/near-optimally without visiting the recharging stations several times. The obtained results for both of the cases are similar, with a slight decrease in the solution cost on Uber instances when allowing multiple visits. In this case, allowing multiple visits cannot improve the results as vehicles do not need to recharge or visit a recharging station several times to fulfill the energy requirement at the end of the route.

5.5.2. Medium-energy restriction scenario: $\gamma = 0.4$

We further conduct experiment with setting a medium-level energy restriction at the end of the route. The experiment results are summarized in the following Table.

Table 9

Results comparison: multiple visits v.s. single visit under $\gamma = 0.4$

$\gamma = 0.4$	Multiple visits	Single visit	Gap%	$\gamma = 0.4$	Multiple visits	Single visit	Gap%
Instances	BC(min)	BC(min)	Obj	Instances	BC(min)	BC(min)	Obj
a2-16	237.38	237.38	0	u2-16	57.65	57.65	0
a2-20	280.70	280.70	0	u2-20	56.34	56.34	0
a2-24	**348.92****	350.68	-0.50%	u2-24	91.63	91.63	0
a3-18	236.82	236.82	0	u3-18	50.74	50.74	0
a3-24	274.80	274.80	0	u3-24	**67.56****	67.96	-0.59%
a3-30	414.06	413.93	0.03%	u3-30	76.99	76.75	0.31%
a3-36	**481.40****	487.58	-1.27%	u3-36	105.62	105.62	0
a4-16	222.49	222.49	0	u4-16	53.58	53.58	0
a4-24	**311.03****	312.33	-0.42%	u4-24	90.95	90.94	0.01%
a4-32	**396.10****	396.45	-0.09%	u4-32	99.64	99.29	0.35%
a4-40	453.84	453.84	0	u4-40	**134.38****	135.92	-1.13%
a4-48	**560.51****	569.87	-1.64%	u4-48	**149.87****	NA	NA
a5-40	**419.83****	419.98	-0.04%	u5-40	123.98	122.57	1.15%
a5-50	**568.90****	574.02	-0.89%	u5-50	**145.69****	146.51	-0.56%
Avg			-0.34%				NA[a]

** Double stars indicate by allowing multiple visits, we find lower-cost solution or solve new instances.

[a] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

It can be observed that, allowing multiple visits to the recharging station can: (1) further decrease the solution cost, (2) solve unsolved instance (i.e., u4-48). By carefully comparing the routing result of each instance, 17.9% instances visit a recharging station multiple times.

5.5.3. High-energy restriction scenario: $\gamma = 0.7$

We also analyze the effect of allowing multiple visits to the recharging station with high-energy restriction. The experiment results are summarized in the following Table.

Table 10

Results comparison: multiple visits v.s. at most one visit under $\gamma = 0.7$

$\gamma = 0.7$	Multiple visits	Single visit	Gap%	$\gamma = 0.7$	Multiple visits	Single visit	Gap%
Instances	BC(min)	BC(min)	Obj	Instances	BC(min)	BC(min)	Obj
a2-16	240.66	240.66	0	u2-16	**58.47****	59.19	-1.22%
a2-20	**281.72****	NA	NA	u2-20	56.86	56.86	0
a2-24	**350.39****	364.19	-3.79%	u2-24	NA	NA	NA
a3-18	**240.04****	240.58	-0.22%	u3-18	50.99	50.99	0
a3-24	**276.27****	281.82	-1.97%	u3-24	**68.39****	68.98	-0.86%
a3-30	**421.39****	NA	NA	u3-30	**78.28****	78.83	-0.70%
a3-36	**494.27****	NA	NA	u3-36	**105.79****	107.13	-1.25%
a4-16	**222.49****	223.13	-0.29%	u4-16	53.87	53.87	0
a4-24	**316.79****	318.29	-0.47%	u4-24	**91.69****	92.11	-0.46%
a4-32	**395.79****	427.92	-7.51%	u4-32	**99.50****	100.53	-1.02%
a4-40	**461.54****	NA	NA	u4-40	**153.15****	NA	NA
a4-48	**576.57****	NA	NA	u4-48	NA	NA	NA
a5-40	**422.00****	434.49	-2.87%	u5-40	**128.79****	NA	NA
a5-50	**583.28****	625.95	-6.82%	u5-50	145.80	144.75	0.73%
Aver			NA[a]				NA[a]

** Double stars indicate by allowing multiple visits, we find lower-cost solution or solve new instances.

[a] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

It is clear from the above Table that allowing multiple visits to recharging stations can efficiently decrease the total cost. Moreover, all the unsolved instances in the Cordeau instance set can be solved when the visits to recharging stations are not limited. As for the Uber instance set, two unsolved instances can be solved. The Uber instances are easier to be solved than the Cordeau instances because the number of recharging stations associated in each instance is larger than the Cordeau instances. The Cordeau instance set thus has a larger improvement compared to the Uber instance set while relaxing the visits to the recharging station. In the case of high-energy-level constraint (i.e., $\gamma = 0.7$), 20 out of 28 instances have been further solved/improved while allowing multiple visits to the recharging station.

5.6. Results on Large-scale E-ADARP Instances

In the existing literature ([9]), E-ADARP can be solved by up to 5 vehicles and 40 requests. We also employ the proposed algorithm to solve the newly-introduced Ropke instances with up to 8 vehicles and 96 requests. We present the results that limit the visit to recharging station in the first part. Then, we relax the visit to the recharging stations and present the corresponding results in the second part. For each part, three different energy-level restrictions are considered.

5.6.1. Case 1: Visit the recharging station at most once

The results of three energy restrictions with limiting the visit to recharging station are shown below:

Table 11

Algorithm results on Ropke instances with recharging station being visited at most once

Instances	BC(min)	AC(min)	AT(s)	Instance	BC(min)	AC(min)	AT(s)
$\gamma = 0.1$				$\gamma = 0.4$			
a5-60	705.38	722.33	419.33	a5-60	NA	NA	609.37
a6-48	520.94	529.79	186.56	a6-48	517.80	527.08	214.14
a6-60	704.93	720.21	290.38	a6-60	737.72	771.55	363.87
a6-72	807.68	827.18	446.28	a6-72	NA	NA	791.96
a7-56	625.24	641.98	179.24	a7-56	627.20	638.90	201.74
a7-70	788.84	802.99	301.64	a7-70	867.64	867.64	397.85
a7-84	920.38	930.79	483.89	a7-84	NA	NA	621.06
a8-64	653.39	671.46	203.37	a8-64	663.37	680.57	225.18
a8-80	841.99	861.02	317.50	a8-80	874.98	915.81	486.12
a8-96	1107.73	1130.18	489.74	a8-96	NA	NA	817.27

When adding a high-energy restriction ($\gamma = 0.7$) on the Ropke instance set, none of the instances can be solved. It is due to the limited recharging station associated for each instance (only 3 recharging stations). With the increasing size of the problem, when all the recharging stations have been visited by vehicles, the other vehicles cannot return to the depot with enough energy. We then conduct experiments on allowing unlimited visits to the recharging stations.

5.6.2. Case 2: Visit the recharging station multiple times

The results of three energy restrictions without limiting the visit to recharging station are shown as the following Tables. From the results, when the multiple visits to the recharging stations are allowed, all the instances can be solved when $\gamma = 0.7$ and $\gamma = 0.4$. The advantages of considering multiple visits in the model have been proved.

Table 12

Algorithm results on Ropke instances with recharging station being visited multiple times

Instances	BC(min)	AC(min)	AT(s)	Instance	BC(min)	AC(min)	AT(s)
$\gamma = 0.1$				$\gamma = 0.4$			
a5-60	707.57	725.72	410.93	a5-60	714.05	725.96	550.53
a6-48	519.83	529.41	182.12	a6-48	516.99	525.02	202.55
a6-60	707.49	715.60	279.29	a6-60	706.96	716.64	327.04
a6-72	795.41	819.97	439.66	a6-72	803.03	825.93	607.21
a7-56	625.94	644.53	174.76	a7-56	629.01	642.75	200.44
a7-70	777.71	805.00	289.66	a7-70	791.28	812.34	351.47
a7-84	909.36	931.47	459.66	a7-84	925.36	948.58	557.85
a8-64	659.11	678.24	191.40	a8-64	659.53	669.54	212.03
a8-80	851.22	867.19	313.05	a8-80	850.64	864.74	411.64
a8-96	1099.64	1139.16	483.55	a8-96	1113.15	1142.23	688.69

Table 13

Algorithm results on Ropke instances with recharging station being visited multiple times

Instances	BC(min)	AC(min)	AT(s)
$\gamma = 0.7$			
a5-60	736.06	742.80	1196.07
a6-48	537.18	542.60	405.64
a6-60	714.01	728.79	634.82
a6-72	827.22	840.25	1476.84
a7-56	639.70	662.17	472.40
a7-70	819.06	836.35	759.50
a7-84	941.18	964.68	997.01
a8-64	679.73	685.55	430.71
a8-80	856.71	886.80	859.40
a8-96	1146.49	1159.15	1393.42

6. Conclusion and Extension

The E-ADARP is a new variant of DARP that differs from the classic DARPs in terms of energy constraints, requiring the impact of visiting recharging stations and the energy requirement to be considered. E-ADARP is

thus more complex. This paper proposed a improved deterministic annealing meta-heuristic that is capable of solving medium to large-sized E-ADARP. The meta-heuristic is enhanced by efficient local search operators. A bi-directional insertion algorithm is integrated to handle the recharging station positions and recharging duration when necessary. In numerical experiments, we first prove the effectiveness and accuracy of proposed algorithm under different scenarios (low-energy, medium-energy, and high-energy restrictions) with comparing to the exact results in previous literature ([9]). The average gap between the algorithm results and the exact results is 0.58% and we have found new solutions on both solved and unsolved instances. Considering the practical world, we also analyze the situation where the multiple visits to the recharging station are allowed. The results demonstrate that in this case, the solution quality can be further improved and most unsolved instances can be solved. As the exact method proposed in the previous literature cannot solve large-scale instances, we develop new instances based on [11], adding problem-related features and make it suitable for our problem. We therefore present the new solutions obtained on large-scale instances under different energy restrictions.

These results offer several new perspectives. The E-ADARP model may be improved to take into account more real-life characteristics. Firstly, time-dependent travel times may occur, especially with traffic jams in peak hours. Secondly, the objective function may consider users' convenience with less waiting or travel times which may be conflicting with the global financial optimization. Operators of our DA heuristic may be extended to consider additional constraints. Having several objective functions, a perspective is to design population multi-objective meta-heuristics, like evolutionary algorithms, to handle conflicting objectives. Our local search operators may be re-used inside evolutionary algorithms to provide mutation operators. Lastly, the nature of the problem is strongly related to dynamic optimization, with new requests to serve or modifications due to uncertainties like traffic jams. Having quick and efficient heuristic algorithms for dynamic E-ADARP is there a crucial issue, where meta-heuristics are promising.

7. Appendix A. Algorithm results comparing with the exact method results

7.1. Low-energy restriction: $\gamma = 0.1$

Table 14 and Table 15 show the results of proposed DA + LS algorithm and the three-index and two-index exact model on adapted Cordeau and Uber instances, respectively. In this scenario, the vehicles should return to the depots with at least 10% of battery capacity. The bold numbers indicate the algorithm found the best-known solutions.

Table 14

Results comparison on Cordeau instances with $\gamma = 0.1$

Instance	DA + LS algorithm		Three-index model[9][a]		Two-index model[9][a]		Gap%	
$\gamma = 0.1$	BC(min)	AT(s)	Obj(min)	CPU(s)	Obj(min)	CPU(s)	3index	2index
a2-16	**237.38**	115.53	237.38*	7.2	237.38*	1.2	0	0
a2-20	**279.08**	247.73	279.08*	400.2	279.08*	4.2	0	0
a2-24	**346.21**	342.33	346.21*	154.8	346.21*	9.0	0	0
a3-18	**236.82**	68.09	236.82*	442.2	236.82*	4.8	0	0
a3-24	**274.81**	138.15	274.81*	1273.8	274.81*	13.80	0	0
a3-30	**413.27**	229.28	413.27	7200	413.27*	102	0	0
a3-36	481.40	342.08	481.72	7200	481.17*	106.80	-0.07 %	0.05 %
a4-16	**222.49**	36.90	222.49*	3195	222.49*	3.6	0	0
a4-24	**310.84**	75.26	310.84	7200	310.84*	31.2	0	0
a4-32	395.36	147.06	413.02	7200	393.96*	612	-4.28 %	0.36%
a4-40	**453.84**	254.10	NA	7200	453.84*	517.2	NA	0
a4-48	563.45	417.86	NA	7200	554.54	7200	NA	1.61 %
a5-40	418.23	164.16	490.49	7200	414.51*	1141.8	-14.73 %	0.90 %
a5-50	571.49	269.47	NA	7200	559.17	7200	NA	2.20 %
Avg		203.43		4505.23		1210.54	NA[b]	0.37 %

[a] Results on a 3.6 GHz Intel(R) Core(TM) with 16 Gb of RAM.

[b] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

The detailed results of algorithm on Uber instances when $\gamma = 0.1$ are shown as below:

Table 15

Results comparison on Uber instances with $\gamma = 0.1$

Instance	DA + LS algorithm		Three-index model[9][a]		Two-index model[9][a]		Gap%	
$\gamma = 0.1$	BC(min)	AT(s)	Obj(min)	CPU(s)	Obj(min)	CPU(s)	3index	2index
u2-16	**57.61**	195.84	57.61*	268.2	57.61*	21	0	0
u2-20	**55.59**	406.69	55.59*	19.20	55.59*	9.6	0	0
u2-24	**91.27**	341.08	91.27	7200	91.27*	432	0	0
u3-18	**50.74**	88.13	50.74*	630	50.74*	10.8	0	0
u3-24	68.05	185.40	68.78	7200	67.56*	130.2	-1.06%	0.73 %
u3-30	**76.75**	373.17	76.99	7200	76.75*	438	-0.31 %	0
u3-36	105.50	585.38	109.12	7200	104.04*	1084.8	-3.32%	1.40%
u4-16	**53.58**	49.59	53.58*	6716.4	53.58*	48	0	0
u4-24	90.72	81.57	89.83*	3476.4	89.83*	13.2	0.99%	0.99%
u4-32	99.77	192.02	NA	7200	99.29*	1158.6	NA	0.48%
u4-40	134.17	290.49	136.93	7200	133.11*	185.4	-2.02%	0.80%
u4-48	149.73	622.96	NA	7200	148.30	7200	NA	0.96%
u5-40	123.57	186.91	NA	7200	121.86	7200	NA	1.40%
u5-50	146.68	344.70	NA	7200	143.10	7200	NA	2.50%
Avg		281.71		5422.16		1280.83	NA[b]	0.69 %

[a] Results on a 3.6 GHz Intel(R) Core(TM) with 16 Gb of RAM.

[b] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

7.2. Medium-energy restriction: $\gamma = 0.4$

Table 16 and Table 17 show the results of proposed DA + LS algorithm and the three-index and two-index exact model on adapted Cordeau and Uber instances, respectively. In this scenario, the vehicles should return to the depots with at least 40% of battery capacity. The bold numbers indicate the algorithm found the best-known solutions.

Table 16

Results comparison on Cordeau instances with $\gamma = 0.4$

Instance	DA + LS algorithm		Three-index model[9][a]		Two-index model[9][a]		Gap%	
$\gamma = 0.4$	BC(min)	AT(s)	Obj(min)	CPU(s)	Obj(min)	CPU(s)	3index	2index
a2-16	**237.38**	144.81	237.38*	7.8	237.38*	1.8	0	0
a2-20	**280.70**	561.60	280.70*	358.2	280.70*	49.8	0	0
a2-24	350.68	427.36	348.04*	252	348.04*	25.2	0.76 %	0.76%
a3-18	**236.82**	76.70	236.82*	285.6	236.82*	4.2	0	0
a3-24	**274.80***	160.60	274.80*	1973.4	274.81*	16.8	0	-0.003%
a3-30	413.93	290.61	413.80	7200	413.37*	99	0.03%	0.14 %
a3-36	487.58	397.12	489.99	7200	484.14*	306.6	-0.49 %	0.71 %
a4-16	**222.49**	40.34	222.49*	3192	222.49*	5.4	0	0
a4-24	312.33	98.01	311.03	7200	311.03*	39.6	0.42%	0.42%
a4-32	396.45	171.15	394.32	7200	394.26*	681.6	0.54%	0.56%
a4-40	**453.84**	302.74	NA	7200	453.84*	417.6	NA	0.38%
a4-48	569.87	521.46	NA	7200	554.60	7200	NA	2.75 %
a5-40	419.98	182.89	454.81	7200	414.51*	1221	-7.66%	1.32%
a5-50	574.02	342.69	NA	7200	560.50	7200	NA	2.41 %
Avg		265.58		4547.79		1233.47	NA[b]	0.64%

[a] Results on a 3.6 GHz Intel(R) Core(TM) with 16 Gb of RAM.

[b] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

* Numbers in bold with star indicate new solution found by proposed algorithm.

The detailed results when we test our algorithm on adapted Uber instances with $\gamma = 0.4$ are shown as below:

Table 17

Results comparison on Uber instances with $\gamma = 0.4$

Instance	DA + LS algorithm		Three-index model[9][a]		Two-index model[9][a]		Gap%	
$\gamma = 0.1$	BC(min)	AT(s)	Obj(min)	CPU(s)	Obj(min)	CPU(s)	3index	2index
u2-16	**57.65**	277.02	57.65*	229.8	57.65*	25.8	0	0
u2-20	**56.34**	544.79	56.34*	83.4	56.34*	12	0	0
u2-24	**91.63**	399.40	NA	7200	91.63*	757.2	0	0
u3-18	**50.74**	106.29	50.74*	772.8	50.74*	13.8	0	0
u3-24	67.96	226.28	67.77	7200	67.56*	220.8	0.28%	0.59%
u3-30	**76.75**	435.40	78.15	7200	76.75*	336.6	-1.79%	0
u3-36	105.62	654.33	NA	7200	104.06*	2010	NA	1.50%
u4-16	**53.58**	66.85	53.58*	5297.4	53.58*	44.4	0	0
u4-24	90.94	101.99	89.83*	5133	89.83*	28.2	1.24%	1.24%
u4-32	**99.29**	229.43	NA	7200	99.29*	2667.6	NA	0
u4-40	135.92	334.47	NA	7200	133.91*	2653.2	NA	1.50%
u4-48	NA	539.37	NA	7200	NA	7200	NA	NA
u5-40	122.57	236.48	NA	7200	122.23	7200	NA	0.28%
u5-50	146.51	395.70	NA	7200	143.14	7200	NA	2.35%
Avg		324.84		5451.17		2169.25	NA	0.57%

[a] Results on a 3.6 GHz Intel(R) Core(TM) with 16 Gb of RAM.

[b] NA indicates the gap cannot be calculated due to the unequal number of solved instances.

References

[1] D. Schrank, T. Lomax, B. Eisele, 2012 urban mobility report, Texas Transportation Institute,[ONLINE]. Available: http://mobility. tamu. edu/ums/report (2012).

[2] W. Feng, M. Figliozzi, An economic and technological analysis of the key factors affecting the competitiveness of electric commercial vehicles: A case study from the usa market, Transportation Research Part C: Emerging Technologies 26 (2013) 135–145.

[3] C. Prins, A simple and effective evolutionary algorithm for the vehicle routing problem, Computers & operations research 31 (12) (2004) 1985–2002.

[4] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Computers & operations research 34 (8) (2007) 2403–2435.

[5] N. Jozefowiez, F. Semet, E.-G. Talbi, Multi-objective vehicle routing problems, European journal of operational research 189 (2) (2008) 293–309.

[6] N. Jozefowiez, F. Semet, E.-G. Talbi, An evolutionary algorithm for the vehicle routing problem with route balancing, European Journal of Operational Research 195 (3) (2009) 761–769.

[7] K. Ghoseiri, S. F. Ghannadpour, Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm, Applied Soft Computing 10 (4) (2010) 1096–1107.

[8] A. M. Altabeeb, A. M. Mohsen, A. Ghallab, An improved hybrid firefly algorithm for capacitated vehicle routing problem, Applied Soft Computing 84 (2019) 105728.

[9] C. Bongiovanni, M. Kaspi, N. Geroliminis, The electric autonomous dial-a-ride problem, Transportation Research Part B: Methodological 122 (2019) 436–456.

[10] J.-F. Cordeau, A branch-and-cut algorithm for the dial-a-ride problem, Operations Research 54 (3) (2006) 573–586.

[11] S. Ropke, J.-F. Cordeau, G. Laporte, Models and branch-and-cut algorithms for pickup and delivery problems with time windows, Networks: An International Journal 49 (4) (2007) 258–272.

[12] J.-F. Cordeau, G. Laporte, The dial-a-ride problem: models and algorithms, Annals of operations research 153 (1) (2007) 29–46.

[13] S. N. Parragh, K. F. Doerner, R. F. Hartl, A survey on pickup and delivery problems, Journal für Betriebswirtschaft 58 (2) (2008) 81–117.

[14] S. N. Parragh, Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem, Transportation Research Part C: Emerging Technologies 19 (5) (2011) 912–930.

[15] S. N. Parragh, J.-F. Cordeau, K. F. Doerner, R. F. Hartl, Models and algorithms for the heterogeneous dial-a-ride problem with driver-related constraints, OR spectrum 34 (3) (2012) 593–633.

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

[17] M. A. Masmoudi, K. Braekers, M. Masmoudi, A. Dammak, A hybrid genetic algorithm for the heterogeneous dial-a-ride problem, Computers & operations research 81 (2017) 1–13.

[18] M. A. Masmoudi, M. Hosny, E. Demir, K. N. Genikomsakis, N. Cheikhrouhou, The dial-a-ride problem with electric vehicles and battery swapping stations, Transportation research part E: logistics and transportation review 118 (2018) 392–420.

[19] S. Erdoğan, E. Miller-Hooks, A green vehicle routing problem, Transportation research part E: logistics and transportation review 48 (1) (2012) 100–114.

[20] M. Schneider, A. Stenger, D. Goeke, The electric vehicle-routing problem with time windows and recharging stations, Transportation Science 48 (4) (2014) 500–520.

[21] D. Goeke, M. Schneider, Routing a mixed fleet of electric and conventional vehicles, European Journal of Operational Research 245 (1) (2015) 81–99.

[22] G. Hiermann, J. Puchinger, S. Ropke, R. F. Hartl, The electric fleet size and mix vehicle routing problem with time windows and recharging stations, European Journal of Operational Research 252 (3) (2016) 995–1018.

[23] G. Hiermann, R. F. Hartl, J. Puchinger, T. Vidal, Routing a mix of conventional, plug-in hybrid, and electric vehicles, European Journal of Operational Research 272 (1) (2019) 235–248.

[24] G. Dueck, T. Scheuer, Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, Journal of computational physics 90 (1) (1990) 161–175.

[25] K. Braekers, A. Caris, G. K. Janssens, Integrated planning of loaded and empty container movements, OR spectrum 35 (2) (2013) 457–478.

[26] O. Bräysy, W. Dullaert, G. Hasle, D. Mester, M. Gendreau, An effective multirestart deterministic annealing metaheuristic for the fleet size and mix vehicle-routing problem with time windows, Transportation Science 42 (3) (2008) 371–386.

[27] S. N. Parragh, K. F. Doerner, R. F. Hartl, Variable neighborhood search for the dial-a-ride problem, Computers & Operations Research 37 (6) (2010) 1129–1138.

[28] T. Vidal, T. G. Crainic, M. Gendreau, C. Prins, A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows, Computers & operations research 40 (1) (2013) 475–489.