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

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

Abstract

This paper studies the Electric Autonomous Dial-a-Ride Problem (E-ADARP) which consists of routing electric autonomous vehicles to transport users from specific origins to specific destinations within predefined time windows. E-ADARP was previously solved for medium size instances, up to 5 vehicles and less than 50 requests with an exact branch-and-cut algorithm. To solve larger instances, this paper proposes a Deterministic Annealing (DA) local search meta-heuristic for E-ADARP using several neighborhoods and repair operators. Highly constrained instances occur with a limitation on recharging stations and with a minimum energy threshold parameter when returning to the depots. Computational results are analyzed, comparing them on instances with three different energy threshold values in order to examine the algorithm performance on loosely to highly constrained instances. On challenging highly constrained instances, our enhanced DA meta-heuristic consistently finds solutions of good quality. Several new best solutions are found on previously solved and unsolved instances, and on new benchmark instances with up to 8 vehicles and 96 requests. Finally, it is shown that allowing multiple visits to recharging stations is helpful to improve feasibility issues and the quality of solutions found. Perspectives of these results are discussed, especially using more realistic recharging constraints and solving multi-objective extensions.

Keywords: Operations Research, Optimization, Dial-a-Ride Problem, Electric Autonomous Vehicles, Rich Vehicle Routing problems, Meta-heuristic, Local Search, Deterministic Annealing

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]. Increasing emissions of greenhouse gases worldwide leads to climate change and the rise of sea levels. To address these concerns, two effective changes to motorized individual transport are considered: using clean energy powered vehicles and providing ride-sharing services. In replacing conventional unsustainably-fueled vehicles with electric powered ones, the rechargeable battery as a power source has the advantages of zero greenhouse gas emission, less energy cost per

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

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 electric autonomous vehicles (EAVs) for user-specific transport requirements.

Ride-sharing services are the second possible research direction being investigated. Ride-sharing services can reduce the number of vehicles en route, rendering transport more efficient, benefiting customers with discounted ride prices. The Dial-a-Ride Problem (DARP) corresponds to a class of combinatorial optimization problems, introduced by [3], 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-windows, capacity of vehicles, maximum user ride time). Extensive studies have been conducted on DARP, using exact methods and meta-heuristics [4, 5]. Studies that combine electric vehicles with DARP are rare. To the best of our knowledge, the only work explicitly considering the use of EAVs in the context of DARP, denoted E-ADARP, is the work of [6], where a branch-and-cut (B&C) algorithm is proposed to solve E-ADARP instances with up to 5 vehicles and 50 requests. The limitation of this study is that the proposed exact method cannot solve medium-to-large-sized instances in a given amount of computational time. In this paper, we seek to solve larger instances of E-ADARP with meta-heuristics.

A challenging issue is here that reference instances for E-ADARP may be highly constrained, with energy levels and recharging constraints, in addition to time window and capacity constraints. Highly constrained instances of optimization problems are a known major bottleneck of meta-heuristics, they induce difficulties to consistently find feasible solutions and they may induce local minima of a poor quality [7]. This phenomenon was studied for the class of vehicle routing problems (VRP), denoted as rich VRP optimization problems with many real-life constraints [8, 9]. Electric vehicles induce a rich VRP with electricity levels and recharging constraints. We note a recent trend to unify formulations and dedicated meta-heuristics for broad classes of VRPs [10]. Especially, Heterogeneous Fleet VRPs contain site dependent and periodic VRPs [11]. However, pick-up and delivery VRP, considering like DARP pick-up and drop-off nodes, are not considered in such unified heuristics for VRPs. Indeed, local moves for pick-up and delivery VRP and DARP have to consider both pick-up and drop-off to guarantee that deliveries are realized by the same vehicle after the pick-up [12].

The contribution of this paper is threefold. Firstly, we design a meta-heuristic approach, namely Deterministic Annealing (DA), to solve E-ADARP with problem-tailored neighborhoods and repair operators. Secondly, extensive numerical experiments are conducted on benchmark instance sets for E-ADARP. Several new best solutions are found on previously solved and unsolved instances, and on new benchmark instances with up to 8 vehicles and 96 requests. Statistical and comparative analyses of the performance of our DA local search on loosely to highly constrained instances illustrate the difficulty of tackling highly constrained instances. Thirdly, we investigate the effect of allowing multiple visits to the recharging station in extension of [6]. The major difficulties for local search induced by highly-constrained instances are lessened considering this more realistic situation, which opens perspectives in the way of modeling constraints for recharging stations.

The remainder of this paper is organized as follows. Section 2 presents an overview of related literature. Section 3 provides the problem definition and notation. The DA meta-heuristic algorithm with E-ADARP-tailored operators

is presented in Section 4. In Section 5, the computational experiments are presented and analyzed to appreciate the performance of our DA meta-heuristic. The paper ends in Section 6 with a summary of the results and contribution of the paper, closing with discussions of the perspectives that are opened by our contributions. To ease the readability of the paper, some result tables are gathered in Appendix A.

2. Literature Review

The DARP has been extensively studied since it was introduced in [3]. Note that a broad class of DARP exists, differing in the constraints, but also in the objective function to model quality of service for customers. Detailed literature reviews on DARP can be found in [4, 5]. According to the problem features of E-ADARP, we focus on reviewing representative works that relate to the following characteristics: heterogeneous vehicle fleets, multiple depots, and the use of electric vehicles. Table 1 summarizes all the representative literature discussed.

Parragh [19] formally introduces the heterogeneous DARP (H-DARP) that considers the needs of different users (i.e., staff, patients) and satisfies their requirements (i.e., normal seat, stretchers, wheelchairs) by assigning different types of vehicles. A Variable Neighborhood Search (VNS) is designed and tested on 36 instances with up to four vehicles and 48 requests. Parragh et al. [20] extend their previous work by considering driver-related constraints. Column Generation (CG) is combined with VNS and yields high-quality solutions for realistic instances. Braekers et al. [21] 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 starts and ends at the same depot. Both an exact method and a meta-heuristic are used in their study, note that DA is introduced for the first time in this study to solve a 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. [21], Masmoudi et al. [22] develop a hybrid Genetic Algorithm (GA) 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. [23] consider using electric vehicles in the H-DARP, and recharging is realized through battery swapping. They use a realistic energy consumption model to formulate the problem, and three enhanced Evolutionary VNS (EVO-VNS) algorithm variants are introduced. Bongiovanni et al. [6] study the electric autonomous DARP (E-ADARP), where the authors consider partial recharging, 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 and a two-index model, and new valid inequalities tailored by problem-specific constraints are introduced in a B&C algorithm. Based on standard instances from [24], 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 optimally. However, no heuristic or meta-heuristic algorithm has been proposed yet for E-ADARP.

We also review some works that propose the approaches to solve VRPs with electric vehicles (e.g., [13], [14], [15], [16, 18]). Among them, Erdoğan and Miller-Hooks [13] first propose 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

Table 1

Representative studies on E-VRP/DARP, comparisons of objective functions, problem characteristics (HF: Heterogeneous Fleet of Vehicles, MD: Multiple Depot, VT: Vehicle Type, and RP: Recharging Policy) exact and heuristic methods to solve the E-VRP/DARP variant

Electric vehicle routing problem related literature							
Studies	**Objectives**	**Exact**	**Heuristic**	**HF**	**MD**	**VT**[a]	**RP**[b]
Erdoğan and Miller-Hooks [13]	distance	MILP	greedy[c]	no	no	AF	FC
Schneider et al [14]	distance	MILP	VNS+Tabu	no	no	EV	FC
Goeke and Schneider [15]	cost	no	ALNS	yes	no	MF	FC
Hiermann et al [16]	cost	B&P	ALNS	yes	no	MF	FC
Desaulnier et al [17]	cost	B&P	no	no	no	EV	FC/PC
Hiermann et al [18]	cost	no	GA+LNS	yes	no	MF	PC
Dial-a-ride problem related literature							
Studies	**Objectives**	**Exact**	**Heuristic**	**HF**	**MD**	**VT**[a]	**RP**[b]
Parragh [19]	bi-objective[d]	B&C	VNS	yes	no	CV	none
Parragh et al [20]	cost	CG	VNS	yes	no	CV	none
Braekers et al [21]	distance	B&C	DA	yes	yes	CV	none
Masmoudi et al [22]	distance	no	hybrid GA	yes	no	CV	none
Masmoudi et al [23]	cost	MILP	EVO-VNS	yes	no	EV	BS
Bongiovanni et al [6]	mixed[e]	B&C	no	no[f]	yes	EV	PC
This present work	mixed[e]	no	DA	no[f]	yes	EV	PC

a VT: Vehicle Type, may be electric vehicles (EV), mixed fleet (MF), conventional vehicles (CV) or alternative fuel (AF)

b RP: Recharging Policy, may be battery swap (BS) or Full Charge (FC) and/or Partial Charge (PC).

c The constructive heuristics include Modified Clarke and Wright Savings and the Density-based clustering heuristics.

d The two objectives of [19] are minimizing routing cost and minimizing user waiting time.

e The objective of [6] is a weighted sum of total travel time and excess user ride time.

f In these DARP models and solving approaches, the fleet of vehicles may he heterogeneous. However, the numerical results are given only with identical vehicles.

traveled distance. However, the capacity restrictions and time window constraints are not considered in their model. Based on this work, Schneider, Stenger, and Goeke [14] propose a more complicated model named E-VRPTW. They extend the work of [13] by using electric vehicles and considering limited vehicle capacity and specified customer time window. 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 [13], 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 [15] 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 (ALNS) using a surrogate function to evaluate violations efficiently. Desaulniers et al.[17] extend the E-VRPTW by considering different recharging policies. Recharging policies are discussed, including full recharging plus single/multiple visits to recharging stations, partial recharging plus single/multiple visits to recharging stations. A bi-directional labeling algorithm is proposed to solve the problem with different recharging policies. Hiermann, Puchinger, Ropke, and Hartl [16] extend the work of [15] by taking into account the heterogeneous aspect (i.e., fleet composition). They solve the problem by ALNS. A labeling algorithm is embedded to optimize the positions of recharging stations. The recharging policy considered in this work is also full recharging with a constant recharging rate. Note that labeling algorithms are also used by [16, 17] for CG algorithms in a Branch-and-Price (B&P) exact algorithm. Hiermann et al. [18] 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 techniques 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, we conclude that a combination of heterogeneous vehicles, multiple depots, electric vehicle fleet, and partial recharging in DARPs has rarely been investigated in the previous literature. However, these problem characteristics often jointly appear in practice, especially with the prevalence of electric vehicles. As Table 1 presents, the only research work that considers all these aspects is [6]. However, the proposed B&C algorithm cannot be applied practically to solve large-scale instances. This paper focuses on filling this gap by introducing an efficient meta-heuristic algorithm that can provide near-optimum solutions within reasonable computational time for large-sized instances.

3. Problem Definition

In this section, we recall the E-ADARP definition proposed in [6]. First, we will consider E-ADARP as is, allowing us to compare our results to the ones from [6] . We will discuss the practical interest and the realism of extending the original problem by allowing multiple visits to recharging stations.

3.1. Notation and problem statement

We report in Table 2 the notations and definitions for the index, sets and parameters.

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.

The depot must 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.

Table 2 E-ADARP problem sets, parameters notations and descriptions

Notations	Descriptions
$P^u = \{1, \cdots, i, \cdots, n\}$	Set of pickup locations
$D^u = \{n+1, \cdots, n+i, \cdots, n\}$	Sets of drop-off locations
$N = P^u \cup D^u$	Set of pickup and drop-off locations
$K = \{1, \cdots, k\}$	Set of available vehicles
P^d	Set of original depots
D^d	Set of all available destination depots
F	Set of recharging stations
$V = N \cup P^d \cup D^d \cup F$	Set of all possible locations
$t_{i,j}$	Travel time from location $i \in V$ to location $j \in V$
e_i	Earliest arrival time window at $i \in V$
l_i	Latest departure time window at $i \in V$
s_i	Service duration at $i \in V$
q_i	Change in load at $i \in N$
m_i	Maximum user ride time for customer at $i \in P^u$
VC_k	The vehicle capacity for vehicle $k \in K$
BC_k	The battery capacity for vehicle $k \in K$
γ	The minimum battery level ratio
α_f	The recharge rate at recharging station $f \in F$

For each vehicle $k \in K$, the maximum capacity for each vehicle is denoted as VC_k. 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 [6], 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 arrival and latest departure 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 travel time on each arc is denoted as $t_{i,j}$. The maximum travel duration for request i is denoted as m_i.

At a recharging station, the amount of energy recharged is 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). Vehicles can be partially recharged while visiting recharging stations and must 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.

Regardless of the features that the electric vehicle fleet inherits, vehicle autonomy needs to be taken into account. The route duration constraints are not included as we do not have to consider the driver's shift.

3.2. Constraints of E-ADARP

To summarize, the E-ADARP consists of the following features that are different from the typical DARPs:

- detour to recharging stations on the route;
- partial recharging at recharging stations;
- vehicle can be located at different origin depots;
- no restriction on route duration time.

The following constraints should be satisfied:

- Every route starts from an origin depot and ends at a destination depot;
- For each request, its corresponding pickup and drop-off node belong to the same route, and the pickup node is visited before its drop-off node;
- User nodes and origin depots are visited once, while each destination depot can be visited at most once;
- The maximum vehicle capacity must be respected at each node;
- Each node is 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;
- The maximum user ride time is not be exceeded for any of the users;
- The battery level at the destination depot must surpass the minimal battery level;
- The energy contained in the vehicle does not exceed the battery capacity;
- The recharging station can only be visited when there is no passenger on-board;
- Each recharging station can only be visited once by one vehicle. (this constraint will be discussed)

Figure 1 presents an example of E-ADARP including 4 vehicles and 16 requests. Each requests contains the pickup node (denoted as $i+$) and the corresponding drop-off node $i-$. If its battery limitation is not satisfied, the vehicle must make a detour to a recharging station before returning back to the depot. Each vehicle can start from a different origin depot and return to a different destination depot. Recharging stations can be visited at most once and no passenger is on-board when recharging as shown in Figure 1.

3.3. Objective function of E-ADARP

To consider the user inconvenience, we set the objective function as a weighted sum of total travel time and total excess user ride time; Interested readers can refer to [6] for detailed mathematical formulation using Mixed Integer Linear Programming (MILP) of E-ADARP.

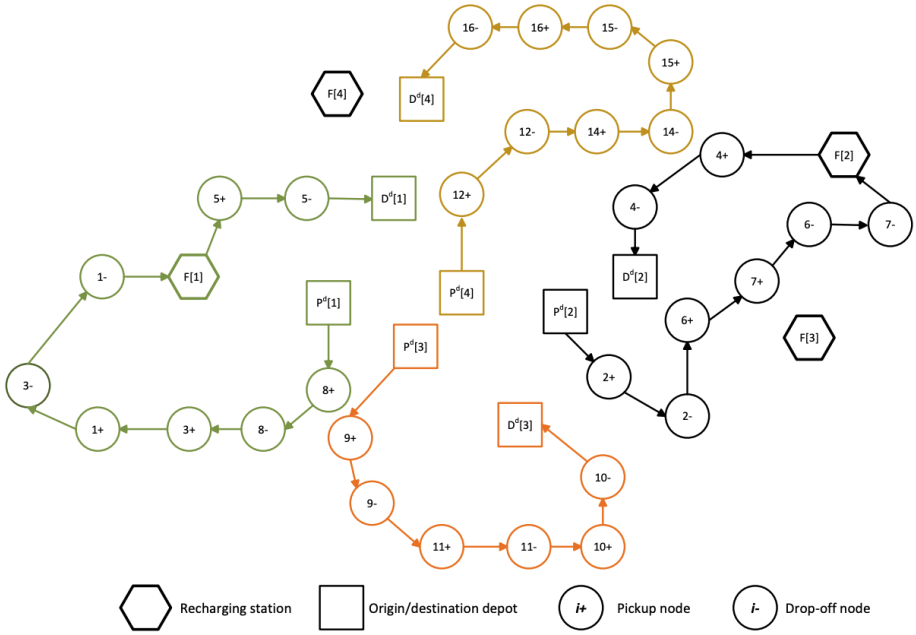

Figure 1 Illustrative example of E-ADARP

3.4. Preprocessing

As mentioned by [24] for DARP, time window tightening is crucial to reduce the problem size, which is beneficial for exact methods, but also for meta-heuristics. Time window tightening allows to remove some arcs from A, by detecting arcs that lead to an infeasible solution,as proposed in [24]:

- for user pickup nodes, e_i is 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 $l_{n+i} = \min\{l_i + m_i + s_i, l_{n+i}\}$.
- for recharging stations, 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;
- 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.

3.5. Multiple visits at recharging stations?

In the E-ADARP instances considered by [6], there are few recharging stations, having different locations. Imposing at most one single visit at recharging station does not seem to be realistic. Real-world constraints would be to have a maximum number of vehicles in recharging stations, time slots for recharging.

It induces that instances of [6] consider recharging station as a scarce resource. It may induce difficulties with high values of γ: it defines a maximal consumption of electricity and few recharging possibilities may be in contradiction with the minimal electricity consumption do serve all the requests.

On the contrary, relaxing this constraint and allowing multiple visits to recharging station still induces a difficult constrained problem. While having maximal time windows, minimal thresholds γ of energy when returning to depots, it is penalizing in terms of cost function and time window feasibility to visit recharging stations often. Relaxing this single visit constraint induces to consider the recharging station as an unlimited resource, which seems more realistic for many applications.

In this paper, our algorithm will be able to handle or to relax this single visit constraint in the E-ADARP framework. Numerical experiments will analyze whether this constraint is crucial for efficiency of optimization algorithms and cost of optimal or near optimal solutions. A numerical issue is to determine how these constraints in recharging stations influence the solving capabilities.

4. Deterministic Annealing Algorithm for E-ADARP

In this section, we design a DA approach to solve E-ADARP with problem-tailored neighborhoods and repair operators. This section is organized as follows. First, the global structure of the algorithm will be presented in section 4.1. Second, a constructive heuristic to generate an initial solution will be explained in section 4.2. Then, the variants of neighborhoods will be presented in section 4.3. Third, we present a method to compute solution cost when obtaining a new solution in section 4.4. Finally, we present a repair operator embedded in the DA local search to insert recharging stations, with a bi-directional insertion algorithm that is embedded in each operator to explicitly (section 4.5).

4.1. Algorithm structure

DA is a variant of simulated annealing and was first introduced by [25]. Recent research shows that DA can obtain near-optimal or optimal solutions for a series of vehicle routing problems ([26, 27]). To the best of our knowledge, the only paper that implements DA to solve DARP is that of [21], The principle of DA local search is as follows: in each iteration, we select randomly a neighboring solution x' from the current solution x. If the obtained cost $f(x')$ is better than the current solution cost $f(x)$, the neighboring 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 ([21]). 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.

The algorithm structure for the proposed DA is presented in Algorithm 1. The algorithm input is the obtained solution x_{init} from a constructive heuristic, and a list of indexed neighborhoods that will be operated iteratively, and parameters of DA local search: a maximal number of iterations Nb_{iter}, the initial and maximal temperature T_{max}, the number of iteration n_{imp} without improvement to restart from the best solution found. It should be mentioned that the initial solution x_{init} shall be feasible for E-ADARP constraints except that only a subset of requests may be served. An empty solution is then acceptable as an initialization of DA local search, section 4.2 gives a better initialization with a parallel insertion heuristic. Serving a partial number of requests, the solution cost is denoted as $c(x)$, and the number of requests served in the solution is updated to Nb_{req} so that a lexicographic optimization considers cost comparison in $c(x)$ values only if it does not worsen the number of requests served.

Algorithm 1 Deterministic Annealing Algorithm for E-ADARP

Input: Initial solution x_{init}, n types of neighborhoods $\mathcal{N}_1, \ldots, \mathcal{N}_n$, and DA parameters: Nb_{iter}, T_{max}, n_{imp},

Output: Best solution found x_b;

Initialization: $T = T_{max}$, $i_{imp} = 0$, $iter = 0$, $x = x_b = x_{init}$, $c(x_b) = c(x) = c(x_{init})$;

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

while $iter \leqslant Nb_{iter}$ **do**
  $i_{imp} \leftarrow i_{imp} + 1$;
  for $j = 1 \rightarrow n$ **do**
    Select randomly a neighboring solution x' from x using neighborhood $\mathcal{N}_j$;
    if x is battery-infeasible solution **then**
      bi-directional insertion algorithm to repair in-feasibility;
    end if
    if $c(x') < c(x) + T$ **then**
      $x \leftarrow x'$;
    end if
  end for
  if $Nb_{req} \leqslant Nb_{user}$ **then**
    try *AddNewRequest* operator to serve more requests
  end if
  $iter \leftarrow iter + 1$
  $Nb'_{req} \leftarrow \sum_{k \in K} \sum_{i \in P^u} v'_{i,k}$
  if $c(x') < c(x_b)$ **and** $Nb'_{req} = Nb_{req}$ **or** $Nb'_{req} > Nb_{req}$ **then**
    $x_b \leftarrow x'$
    $i_{imp} \leftarrow 0$
  end if
  if $i_{imp} > 0$ **then**
    $T \leftarrow T - T_{max}/T_{red}$
    if $T < 0$ **then**
      $r \leftarrow$ random number between 0 and 1
      $T \leftarrow r \times T_{max}$
      if $i_{imp} > n_{imp}$ **then**
        $x \leftarrow x_b$
        $i_{imp} \leftarrow 0$
      end if
    end if
  end if
end while
return x_b

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 an

initial solution x_{init}. 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 restarted 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 solutions (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 presented in section 4.5 is called to insert recharging stations at proper places. Each operator (intra- or inter-route) returns a random neighbor x' from current solution x. 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.

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.

Algorithm 2 Parallel Insertion Heuristic

Input: Initialize the set of non-assigned users sorted in increasing order of earliest pick-up time;
Randomly selected m vehicles and assign the first m users to different vehicles;

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

```
1: repeat
2:     Select the first non-assigned user from the list;
3:     Sort vehicles in increasing order of distance between the last assigned user and the selected user;
4:     Select the first vehicle in the list;
5:     repeat
6:         Check the feasibility of the user's pick-up and drop-off nodes;
7:         if one or more feasible options are found then
8:             Insert the user in the best position;
9:         else
10:            Select the next vehicle in the list;
11:        end if
12:    until the user is inserted;
13:    Remove the user from the list of non-assigned users;
14: until any user cannot be inserted into any routes;
```

4.2. Insertion Heuristic

We use a parallel insertion heuristic to construct an initial solution. We first randomly generate 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 remaining 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 route with the first element in the list is calculated and sorted in increasing order. Then, route by route, the new users will be examined according to their feasibility. If a user can be inserted into a specific route, its pick-up 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. Algorithm 2 explicitly describes the whole insertion process.

4.3. Variants of neighborhoods and local search operators

As in VNS local search, several neighborhoods are used, a local minimum of the multi-neighborhood DA is a local minimum for each type of neighborhoods, it increases the quality of local minima. It motivates the nested choice of neighborhoods in Algorithm 1.

Seven neighborhoods are applied to improve the initial solution generated from the constructive heuristic. Among them, three are intra-route neighborhoods (i.e., *exchange-pick-up*, *exchange-drop-off*, *exchange-two-neighboring-node*), three are inter-route neighborhoods (i.e., *relocate*, *exchange*, and *2-opt*). To tackle the requests that could not be inserted into any routes during the constructive heuristic, the *AddNewRequest* operator is applied in each iteration to try to insert un-served requests into routes.

Exchange pick-up neighborhood aims at swapping the position of two consecutive points (i, j), where point i is a pick-up node and point j is not the corresponding drop-off node. In each iteration, one pick-up node is selected randomly. If the successor of this pick-up node does not correspond to its drop-off node, then the two positions are exchanged. The following figure is an example of applying this neighborhood.

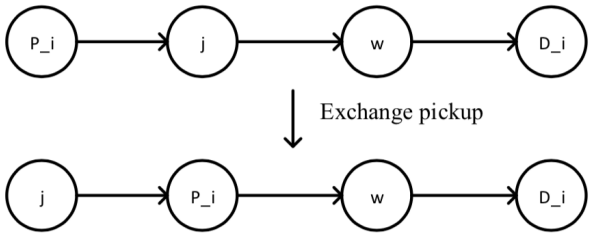

Figure 2 Exchange pick-up neighborhood

Exchange drop-off is designed to swapping the position of two consecutive nodes (i, j), where point j is a drop-off node and point i is not the corresponding pick-up 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 pick-up node, then the two positions are exchanged.

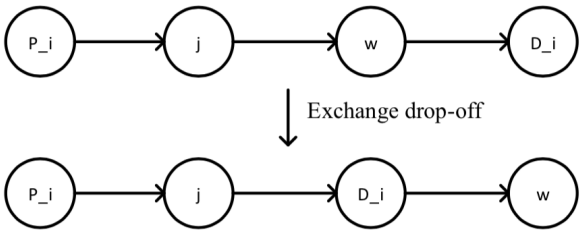

Figure 3 Exchange drop-off neighborhood

There is another situation shown in Figure 4, where the successor of pick-up node P_i is its drop-off D_i, and the predecessor of drop-off node D_j is its corresponding pick-up P_j, but we can still exchange D_i and P_j to obtain a new solution. This operation is realized by *exchange-two-consecutive-nodes*.

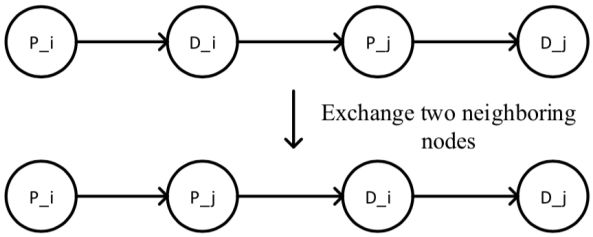

Figure 4 Exchange two consecutive nodes neighborhood

Relocate neighborhood removes a user request from its current route and re-inserts the request at the best position of the route of another vehicle. The neighborhood is applied on a randomly selected user request of a single randomly selected route. It should be mentioned that this neighborhood may eliminate a route if the selected route contains a single request only. When the number of routes activated in the solution is less than the number of available vehicles, it is acceptable to assign a request to an empty vehicle. Figure 5 presents how the *relocate* neighborhood works.

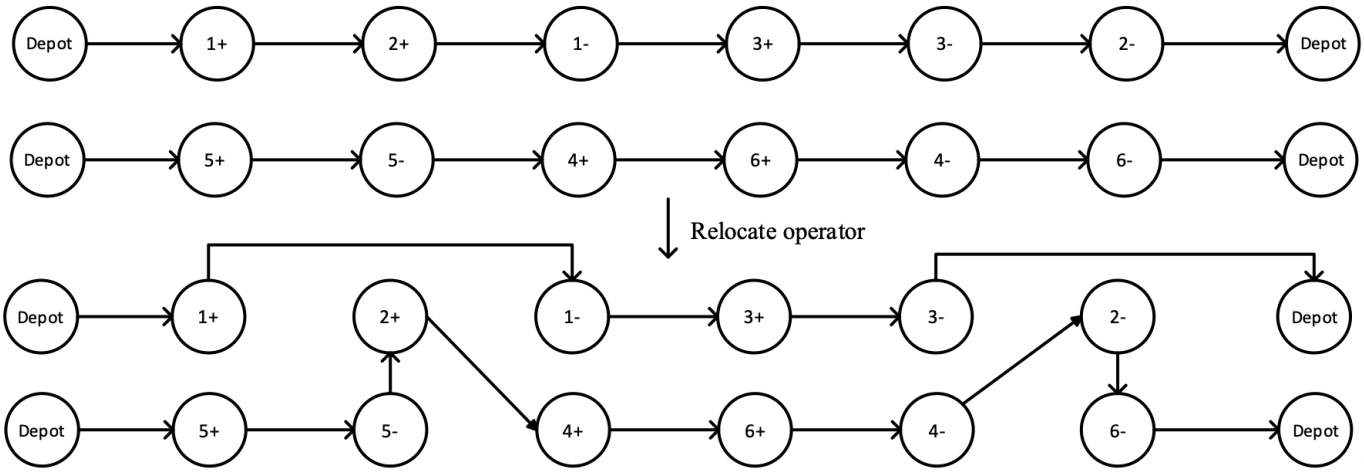

Figure 5 Relocate neighborhood

The *exchange* neighborhood (shown in Figure 6) swaps two requests of two different routes. The pick-up (drop-off) vertex of the first route can only be inserted in the same position as the pick-up (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.

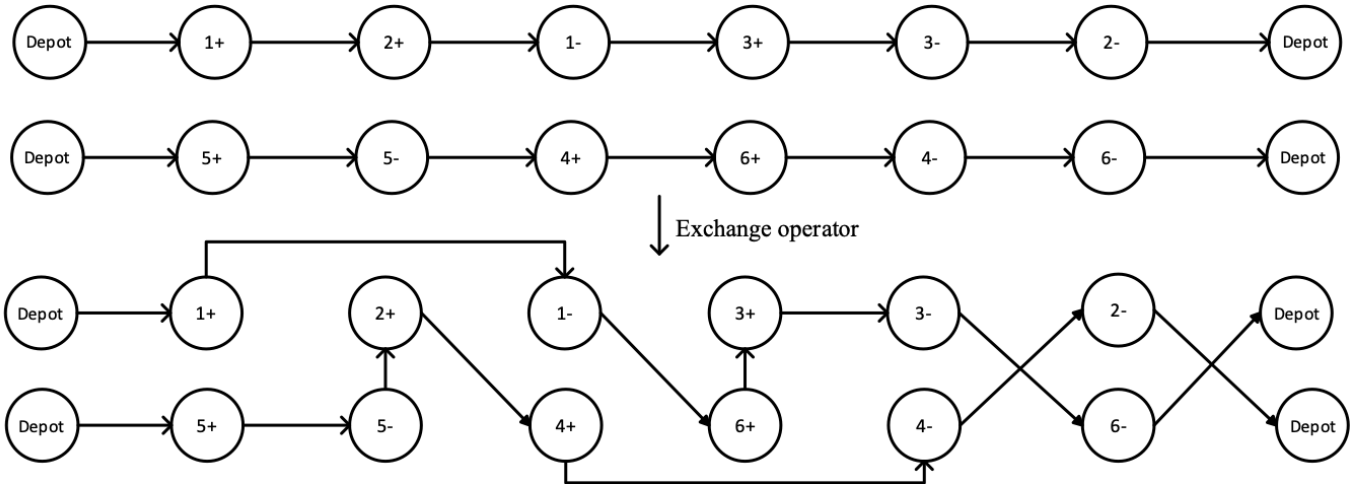

Figure 6 Exchange neighborhood

The 2-opt neighborhood 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 $route_1$. Figure 7 illustrates how the *2-opt* neighborhood works.

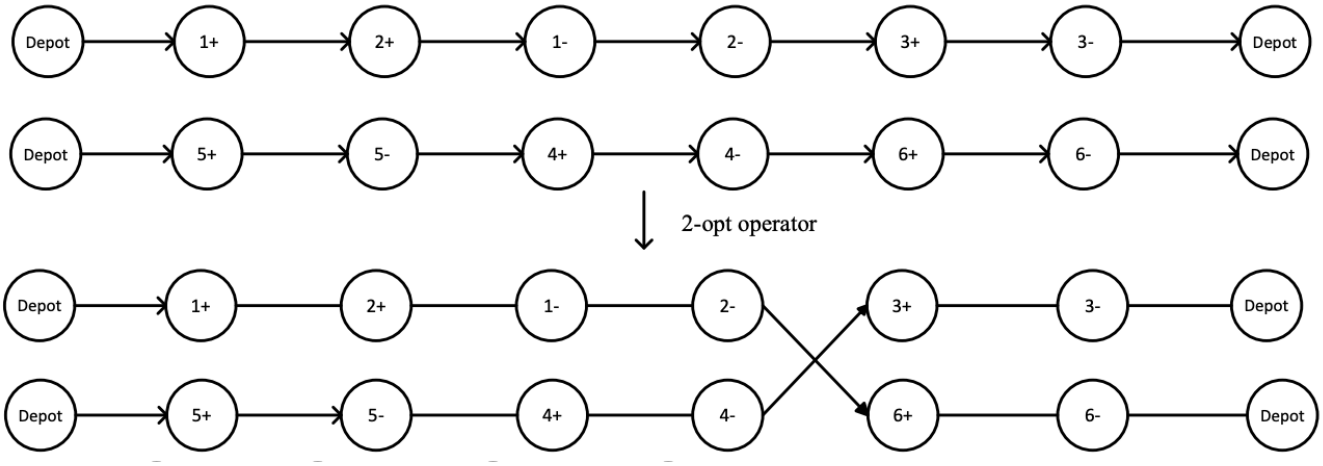

Figure 7 2-opt neighborhood

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. Figure 8 describes how the "rejected" request is added.

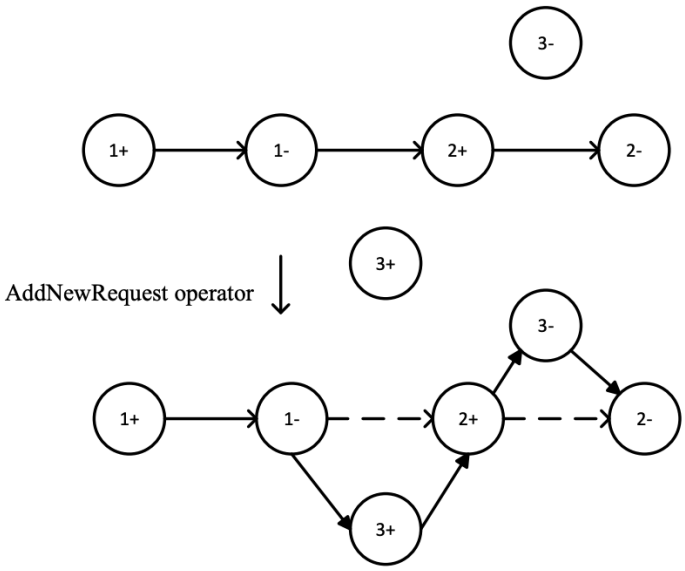

Figure 8 *AddNewRequest* operator

4.4. Route Evaluation

For local search approaches, quick cost computations of neighboring solutions is a crucial element to aggressively explore many feasible solutions in a defined time limit. Cost computation for a neighbor point is quick and straightforward when minimizing only the routing distances as in pick-up and delivery VRPs. For DARPs, other objective functions, like the minimization of user ride time may be more time consuming [12].

Each time when we obtain a new neighboring solution, we need to calculate its solution cost. We use an adapted version of the eight-step evaluation scheme (shown in Algorithm 3) that is introduced by [12]. Differently, we only accept feasible solutions. We borrow the definition of the forward time slack F_i in [12]:

$$F_i = \min_{i \leqslant j \leqslant q} \left\{ \sum_{i < p \leqslant j} W_p + (min\{l_j - B_j, L - P_j\})^+ \right\}$$

W_p denotes the waiting time at node p, with q as the last node on the route. P_j is the ride time of customer whose drop-off vertex is j and j is visited before i. B_j is the service begin time at node j and l_j is the latest time window at node j. F_i is the maximum amount of time can be delayed at node i without violating the time window and user ride time constraint.

4.5. Bi-directional Insertion Algorithm

To repair battery-infeasible solutions, a bi-directional insertion algorithm is embedded in our proposed DA local search. 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 Algorithm 5) and the backward insertion algorithm (shown in Algorithm 6).

In the bi-directional algorithm (Algorithm 4), line 1 to line 7 determine 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

Algorithm 3 Adapted eight-step evaluation scheme

Input: Solution x that has checked time window, capacity, energy constraint;

Output: Solution cost $c(x)$;

Set $D_0 := e_0$;

Compute A_i,W_i,B_i,D_i for each node i in the route;

Compute F_0;

Set $D_0 := e_0 + min(F_0, \sum_{0<p<q} W_p)$;

Update A_i,W_i,B_i,D_i for each node i in the route;

Compute L_i for each request served on the route

if all $L_i \leqslant m_i$ **then**

    for every node j that is a pick-up node **do**

        Record the value of A, B, W, D, L in A', B', W', D', L';

        Compute F_j;

        Set $B_j := B_j + min\{F_j, \sum_{j<p<q} W_p\}$; $D_j := B_j + s_j$.

        Update A_i,W_i,B_i,D_i for each node i after node j in the route;

        Update ride time L_i for each request whose drop-off node is after j;

        if $L_i > L_i'$ **then**

            Keep the value of A, B, W, D, L as A', B', W', D', L';

        end if

    end for

end if

Return Solution cost $c(x) = \sum_{0 \leqslant p \leqslant q-1} t_{p,p+1} + \sum_{0<i<q, i \in P^u} (L_i - t_{i,n+i})$

the vehicle load is zero. Since the order 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 4 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 (Algorithm 5) 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 $RequiredTime$) 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.

Algorithm 5 Forward Insertion Algorithm

Input: Energy negative solution x;

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

1: Calculate last energy-positive node index;
2: **while** end depot energy is negative **do**
3: $RT = max\{0, min\{RequiredTime, AllowedTime\}\}$;
4: **if** RT is not zero **and** there exists reachable recharging stations **then**
5: Inserting the recharging station that has minimum detour;
6: **if** energy constraint satisfied **then**
7: Record $index_{end}$;
8: Update energy list;
9: **else**
10: Recharge RT;
11: Update energy list;
12: Move to the next position;
13: **end if**
14: **else**
15: Solution cannot be fixed, return $\emptyset, \emptyset, \emptyset$;
16: **end if**
17: **end while**
18: **return** x'**,** $index_{end}$, updated energy list;

We use the backward insertion algorithm (Algorithm 6) to repair the minimum battery level infeasibility. Firstly, We reverse 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 6 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; ▷ The possible positions are therefore visited in a backward manner
3: **for** $i = 1 \rightarrow index_{end}$ **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**

It should be mentioned that the recharging station is restricted to be visited at most once in [6]. To be comparable with the benchmark results in [6], we first restrict the recharging station to be visited at most once. Therefore, the recharging station that has been inserted in the solution will not be reconsidered when inserting (corresponding to line 5 in Algorithm 5 and line 6 in Algorithm 6). In Section 5.4, we relax the visits to recharging stations, and then the visited recharging stations are reconsidered to be inserted. Hence, the bi-directional insertion algorithm is made generic to solve standard E-ADARP instances with single visits of recharging stations and to consider the multiple visit relaxation in the DA local search of Algorithm 1.

5. Computational experiments and results

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 an Intel Xeon Gold 6230 20C 2.1GHz computer. This section is organized as follows. The benchmark instances and the experimental design for the computational experiments are introduced in the first part. Then, a sensitivity analysis is conducted to find good parameter settings for the

DA algorithm. After ensuring the robustness of parameters and operators, we analyze the performance of the DA algorithm on standard E-ADARP instances, and allowing multiple visits to recharging stations.

5.1. Instance Characteristics and Experimental Design

To allow reading and analyzing the result tables, this section presents the benchmark instances that are used to test the algorithm performance, their characteristics and the notations for the computational experiments.

5.1.1. Benchmark Instances

Instances are named following the pattern xK-n where $x \in \{c, u, r\}$, K is the number of vehicles and n is the number of requests. Three sets of instances are considered in the experiments, which differentiates $x \in \{c, u, r\}$:

- "c" denotes the standard DARP benchmark instance set from [24] extended with features of electric vehicle and recharging stations by [6]. To simplify, we call them "Cordeau instances". For these Cordeau instances, the number of vehicles is in range $2 \leqslant K \leqslant 5$ and the number of requests is in range $16 \leqslant n \leqslant 50$.
- "u" denotes instances based on the ride-sharing data from Uber Technologies (instance name starts with "u") that were adopted from [6]. To simplify, we call them "Uber instances". For these Uber instances, the number of vehicles is in range $2 \leqslant K \leqslant 5$ and the number of requests is in range $16 \leqslant n \leqslant 50$, as in Cordeau instances.
- "r" denotes larger DARP benchmark instances build from [28] using the same extension rules to have E-ADARP instances from DARP instances of Cordeau. To simplify, we call them "Røpke instances". For these Røpke instances, the number of vehicles is in range $5 \leqslant K \leqslant 8$ and the number of requests is in range $60 \leqslant n \leqslant 96$.

Cordeau instances 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 Røpke instances 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 [6]. The preprocessed data that extract requests information from the raw data provided by Uber Technologies are published on the website (https://luts.epfl.ch/wpcontent/uploads/2019/03/e_ADARP_archive.zip).

For these three datasets, preprocessing presented in section 3.4 applies. If pre-processing techniques are crucial for the efficiency exact methods [29] as used for the B&C results of [6], arc elimination has also an impact for our DA local search when selecting randomly a neighbor points, it avoids trying a neighbor solution that is obviously infeasible.

5.1.2. From loosely to highly constrained instances

The Uber instances differ from Cordeau instances in:

- The pre-processed time windows in Cordeau instances are tighter than that of Uber instances;
- Only three recharging stations are included in each Cordeau instance, while five recharging stations are contained in each Uber instance.

At a first glance, Cordeau instances are more constrained than Uber instances for corresponding instance characteristics. For comparative analyzes, parameter γ allows also to have a graduation from loosely to highly constrained instances among the three datasets.

In each set of instances, we compute results for three different minimum energy threshold ratios when returning to the depot, namely $\gamma = 0.1, \gamma = 0.4, \gamma = 0.7$, as it was already presented in [6]. The higher the value of γ, the stricter the requirement for the energy maintained when vehicles return to the depot. Increasing the value of γ, we keep only a subset of feasible instances, to increases the chances to have unfeasible instances, otherwise it increases the optimal value of the instances. The problem is therefore more difficult to be solved, for an exact method such results were presented by [6], for our DA local search, highly constrained instances are more difficult with less feasible moves. This motivated to design several neighborhoods and reparation operators.

As in [7], results are comparatively analyzed keeping in mind the relation among instance subsets. It allows to analyze the impact of having highly-constrained instances for the performance of the DA algorithm, which is a known bottleneck for the efficiency of local search approaches.

5.1.3. Analysis methodology, statistical indicators

DA has deterministic rules to accept a solution contrary to Simulated Annealing and the sequence of neighborhoods is also deterministic, there still remains a randomized part in the selection of neighboring solutions. Unless indicated otherwise, each experimentation runs 30 computations for each instance with different seeds to appreciate the statistical distribution of the solution quality.

For each instance, we present following absolute values:

- BKS is the Best-known Solution from [6], when available;
- BC is the best-cost solution found by the proposed DA algorithm;
- AC is the average-cost solution found by the proposed DA algorithm over the 30 runs.

To appreciate the distribution of the solution found for the 30 runs, we present solutions as gaps to BKS. Having a solution or an absolute statistical value v on a instance, we compute its gap to BKS with:

$$gap = \frac{|\, v - BKS \,|}{BKS}$$

Note that Røpke instances for E-ADARP are studied here for the first time, AC replace BKS so that we can appreciate the distribution of the values for the different seeds.

For a single instance or a subset of instances, we present following average values to analyze the consistency of DA local search:

- Q1% is the average gap to BKS of the first quartile value over the different runs;
- Q3% is the average gap to BKS of the third quartile value over the different runs;
- AT is the average computational time;
- BC% denotes the gap of BC to BKS;
- AC% denotes the gap of AC to BKS;
- FeasRatio denotes the ratio of feasible solutions found;

5.2. Parameter tuning for the DA local search

The performance of the proposed algorithm depends on several parameters that should be set in advance. We therefore identify in this section robust parameter settings in order to ensure the algorithm performance. Parameter tuning is realized on Cordeau and Uber instances, Røpke instances are not considered for parameter tuning.

DA has deterministic rules to accept a solution contrary to Simulated Annealing and the sequence of neighborhoods is also deterministic, there still remains following parameters:

- Number of iterations Nb_{iter} ;
- Maximum threshold value T_{max} for temperature ;
- Threshold reduction value T_{red};
- 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 initially set to 1.5. For other parameters like threshold reduction value T_{red} and restart parameter n_{imp}, we take the same settings as in [21], setting $T_{red} = 300$ and $n_{imp} = 50$.

5.2.1. Sensitivity analysis and parameter tuning for t_{max}

The sensitivity analysis results for t_{max} under different scenarios (i.e., $\gamma = 0.1, 0.4, 0.7$) are shown in Table 3. As for t_{max}, we test seven values for t_{max}. For each value of t_{max}, we perform ten runs on each instance and iterate the proposed algorithm 5000 times for each run. We consider different levels of energy restrictions ($\gamma = 0.1, \gamma = 0.4, \gamma = 0.7$) to select a robust value of t_{max}. The corresponding results are summarized in Table 3. For each value, the solution gaps between the best/average solutions of the proposed algorithm and BKS over ten runs (i.e., BC% and AC%) are calculated. Q1%, Q3%, and AT are also reported. For the scenario of $\gamma = 0.7$, we compare Feasible%.

From Table 3, in the case of $\gamma = 0.1$, the algorithm performs well under all the settings of t_{max}, among which 0.9 seems to be the best with regards to solution quality and computational efficiency. Other values, such as 1.2 and 1.5, can also be selected as a slight difference is found in the solution quality compared to that of 0.9. When

Table 3

Sensitivity analysis for t_{max} under different γ cases, for Uber and Cordeau instances

t_{max}	0.6	0.9	1.2	1.5	1.8	2.1	2.4
$\gamma = 0.1$							
BC%	0.48%	0.24%	0.32%	0.37%	0.43 %	0.76%	0.62%
AC%	NA	0.82%	1.08%	0.74%	1.26 %	1.65%	1.68 %
Q1%	0.58%	0.44%	0.65%	0.53%	0.81 %	1.19 %	1.10 %
Q3%	1.04%	1.03%	1.22%	1.07%	1.57 %	2.03 %	2.27%
FeasRatio	140/140	140/140	140/140	140/140	140/140	140/140	140/140
AT(s)	307.05	317.07	326.37	300.38	330.42	333.21	333.97
$\gamma = 0.4$							
BC%	0.75 %	0.79%	0.93%	0.64%	0.85%	0.80 %	1.23%
AC%	NA	NA	1.96%	1.38%	1.94%	2.18 %	2.37 %
Q1%	1.11%	1.35%	1.26%	1.41%	1.45%	1.63 %	1.69 %
Q3%	NA	2.01%	2.72 %	1.57%	2.31%	2.75 %	2.94%
FeasRatio	140/140	140/140	140/140	140/140	140/140	140/140	140/140
AT(s)	385.97	411.53	434.27	463.64	466.16	480.74	489.20
$\gamma = 0.7$							
FeasRatio	63/140	66/140	71/140	73/140	70/140	68/140	68/140
AT(s)	372.39	360.59	335.50	369.76	400.16	418.10	437.86

The results are the average values among 10 runs (5000 times per run).

[a] NA indicates there are unsolved instances and the gap cannot be calculated.

γ increases to 0.4, the problem becomes more constrained, and the algorithm with $t_{max} = 0.6, 0.9$ cannot solve all the instances within ten runs. In this case, the algorithm with setting $t_{max} = 1.5$ outperforms the algorithm with other t_{max} settings in terms of solution quality. The problem is highly constrained when it comes to $\gamma = 0.7$, and some instances may not have feasible solutions. From the results, $t_{max} = 1.5$ has the highest proportion of feasible solutions comparing to algorithm with other t_{max} values. Hence, we conclude that $t_{max} = 1.5$ can provide us with good solution quality and acceptable computational time in all the cases. We set $t_{max} = 1.5$ in all the further experiments. For values of T_{red} and n_{imp}, we keep the initial settings, ie $T_{red} = 300$ and $n_{imp} = 50$.

5.2.2. Contribution of local search neighborhoods

As the algorithm largely relies on local search operators, the usefulness of neighborhoods is verified. In this part, we analyze the contribution of local search neighborhoods to improve the solution quality. The effectiveness of each local search operator is analyzed and the results of seven different algorithm configurations are shown in Table 4. In each of these configurations, one operator is excluded from the algorithm, and we run each algorithm configuration ten times, with each run iterating the respective algorithm 5000 times. Cordeau and Uber E-ADARP instances are tested, and we calculate the average solution gap from BKS. Results for different algorithm configurations setting the previously selected parameter values ($t_{max} = 1.5$) are summarized in Table 4, with average solution gaps (BC%,

Table 4

Comparison of statistical indicators of DA performance with all neighborhoods to configurations removing one single type of neighborhood: *exchange pickup* (ExPickUp), *exchange drop-off* (ExDropOff), *exchange consecutive* (ExCons), *relocate*, *exchange* and *2-opt*.

Removing	None	ExPickUp	ExDropOff	ExCons	relocate	exchange	2-opt
$\gamma = 0.1$							
BC%	0.37%	0.41%	0.38%	0.41%	1.08%	0.51%	3.22%
AC%	0.74%	1.21%	1.35%	1.38%	2.29%	1.26%	5.84%
Q1%	0.53%	0.74%	0.88%	0.90%	1.50%	0.98%	4.65
Q3%	1.07%	1.38%	1.72%	1.58%	2.78%	1.46%	7.01%
FeasRatio	140/140	140/140	140/140	140/140	140/140	140/140	140/140
AT(s)	300.38	286.37	288.97	284.5	202.04	271.32	281.08
$\gamma = 0.4$							
BC%	0.64%	1.01%	0.95%	0.98%	1.44%	0.79%	3.75%
AC%	1.38%	1.87%	2.02%	2.08%	NA	1.93%	NA
Q1%	0.96%	1.31%	1.6%	1.46%	2.39%	1.33%	4.86%
Q3%	1.57%	2.18%	2.42%	2.56%	NA	2.36%	7.62%
FeasRatio	140/140	140/140	140/140	140/140	140/140	140/140	140/140
AT (s)	463.64	415.42	414.61	434.6	272.77	412.54	408.53
$\gamma = 0.7$							
FeasRatio	73/140	70/140	71/140	68/140	52/140	73/140	47/140
AT (s)	421.88	308.76	310.91	386.59	235.15	352.71	494.45

The results are the average values among 10 runs (5000 times per run) using Cordeau and Uber instances.

AC%, Q1%, Q3%) and computational time (AT) being reported. For the scenario $\gamma = 0.7$, we report AT and the proportion of feasible solutions generated to the total number of solutions.

We can find that each operator performs very well in improving the solution quality, especially the *2-opt* operator. Additionally, the *relocate* and *2-opt* operator contributes to provide more feasible solutions in the case of $\gamma = 0.4, 0.7$. 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 on the analyzed instances.

5.2.3. Sensitivity Analysis on number of iterations

Firstly, we conduct the sensitivity analysis for the number of iterations Nb_{iter}. To identify a good Nb_{iter} to do further experiments, we conduct experiments with all the energy-level restrictions on all the instances. We test ten values of Nb_{iter}, and report BC%, AC%, Q1%, Q3% over thirty runs on the instances. For the scenario of $\gamma = 0.7$, as different settings of Nb_{iter} result in a different number of feasible solutions, we cannot calculate the solution gaps

Table 5

Statistical comparison of DA performance under different iteration times for all γ values on Uber and Cordeau instances

Nb_{iter}	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Low energy restriction $\gamma = 0.1$										
BC%	0.84%	0.60%	0.46%	0.36%	0.27%	0.24%	0.19%	0.16%	0.14%	0.11%
AC%	2.99%	2.04%	1.62%	1.37%	1.20%	1.07%	0.99%	0.91%	0.85%	0.82%
Q1%	2.03%	1.34%	1.04%	0.81%	0.67%	0.59%	0.53%	0.51%	0.47%	0.46%
Q3%	3.74%	2.52%	2.06%	1.73%	1.52%	1.38%	1.26%	1.19%	1.10%	1.07%
FeasRatio	840/840	840/840	840/840	840/840	840/840	840/840	840/840	840/840	840/840	840/840
AT (s)	74.10	143.43	212.33	281.18	349.90	418.27	486.46	554.88	623.47	691.87
Medium energy restriction $\gamma = 0.4$										
BC%	1.30%	0.83%	0.64%	0.51%	0.43%	0.36%	0.34%	0.34%	0.32%	0.32%
AC%	NA	2.43%	1.97%	1.70%	1.50%	1.40%	1.29%	1.22%	1.16%	1.11%
Q1%	2.56%	1.71%	1.40%	1.19%	1.02%	0.93%	0.86%	0.83%	0.78%	0.74%
Q3%	4.37%	2.96%	2.35%	2.04%	1.82%	1.73%	1.62%	1.55%	1.49%	1.44%
FeasRatio	840/840	840/840	840/840	840/840	840/840	840/840	840/840	840/840	840/840	840/840
AT (s)	76.34	148.90	221.35	293.73	366.16	438.51	510.88	583.42	655.89	728.57
High energy restriction $\gamma = 0.7$										
FeasRatio	349/840	391/840	423/840	431/840	441/840	448/840	453/840	461/840	464/840	466/840
AT (s)	63.52	128.06	193.41	259.25	325.15	391.065	457.38	523.78	590.17	656.62

in this case. Therefore, we compare the ratio of instances with feasible solutions found to the total number of runs in this case. The results are shown in Table 5.

Based on the algorithm results, it can be observed that the values of BC%, AC%, Q1%, Q3% are improved with more iterations. Among ten values of Nb_{iter}, 10000 iterations provide us with the best solution quality. We therefore set N_{iter} to 10000 to conduct experiments. The performance of DA is also demonstrated as small results dispersion is found under all the values of N_{iter}. Moreover, we also notice that the computational time grows linearly with the number of iterations, which is coherent with the design of Algorithm 1. For a given number of iteration, the computation time is constant, it is a computational advantage compared to state-of-art exact methods (i.e., branch-and-cut).

Note that choosing $Nb_{iter} = 7000$ or $Nb_{iter} = 8000$ slightly degrades the performances. With such parameters, the computational time will be decreased. Choosing $Nb_{iter} = 10000$ is more robust, especially keeping in mind the evaluation of larger Røpke instances.

5.3. DA performance on standard E-ADARP Instances

In this section, we present the performance of our DA algorithm after the parametrization from the previous section on standard E-ADARP instances, i.e. we consider that each recharging station is visited at most a single time, for the three datasets.

Table 6

Statistical analysis of DA performance under different energy restrictions for Uber and Cordeau instances, comparison to the BKS from [6]

Low energy restriction ($\gamma = 0.1$)	BC%	AC%	Q1%	Q3%	AT(s)
Cordeau instances	0.14%	0.79%	0.54%	0.99%	645.69
Uber instances	0.08%	0.89%	0.44%	1.25%	738.04
Medium energy restriction ($\gamma = 0.4$)	BC%	AC%	Q1%	Q3%	AT(s)
Cordeau instances	0.41%	1.13%	0.80%	1.38%	653.69
Uber instances	0.23%	1.09%	0.68%	1.50%	803.64

To verify the algorithm's robustness, under each energy restriction setting, we compare the algorithm's results on each instance to BKS for Cordeau instances and Uber instances. Table 6 is a summary of algorithm results on benchmark instance sets (i.e., Cordeau data set and Uber data set) when $\gamma = 0.1, 0.4$, where we report the overall statistical indicators BC%, AC%, Q1%, Q3% and AT for each case. The detailed results on each instance of scenario $\gamma = 0.1$ and $\gamma = 0.4$ can be found in Appendix A. We don't report results of $\gamma = 0.7$ in the Table 6 as it is unfair to calculate BC%, AC%, Q1%, Q3% with an unequal number of solved instances, the detailed results are also in Appendix A for comparing results in some representative instances. Computation times are reported in Table 6, 10000 iterations of DA algorithm is approximately and in average twice faster that the B&C, Tables in Appendix A show the CPU time reported by [6].

Under low-energy restriction, ie $\gamma = 0.1$, the proposed DA algorithm can offer high-quality solutions for both instance sets. The values of BC% are quite small (i.e., 0.14 % and 0.08 %), demonstrating that DA algorithm is capable to find most of the best solutions. Indeed, as shown in Tables of Appendix A, DA solve 10 out of 14 Cordeau instances optimally and 9 out of 14 Uber instances optimally. Apart from BC%, the values of other statistical indicators also show the consistency of the proposed algorithm with small values of statistical indicators AC% and Q3% that are closed to BC%.

Under medium-energy restriction, ie $\gamma = 0.4$, all the statistical values increase compared to $\gamma = 0.1$. It illustrates the difficulty of having more constrained for local search approaches. However, values of AC% and Q3% are still acceptable for more constrained instances, validating the efforts on designing operators to have an efficient local search approach even in highly-constrained cases. All the Cordeau instances can be solved with our algorithm, and 9 out of 14 are optimal. For Uber instances, we found 8 instances solutions. The values of AC% are small on both instance sets (1.13% on Cordeau instance set and 1.09% on Uber instance set).

Table 7 present the new best solution found by our DA algorithm on standard E-ADARP instances, from the result tables in Appendix A.

The B&C algorithm from [6] is very efficient on Uber and Cordeau instances for the loosely constrained and easier instances with $\gamma = 0.1$, with many solution that are proven optimal. For Uber and Cordeau instances with parameter $\gamma = 0.4$, solution found by B&C are still of an excellent quality, we only provided a slight improvement

Table 7

Best results found on Cordeau, Uber and Røpke instances, when improving the previous BKS or being the first feasible solution found

Previous best solutions $\gamma = 0.1$				Current best solutions $\gamma = 0.1$			
Instance	BC(min)	Instance	BC(min)	Instance	BC(min)	Instance	BC(min)
r5-60	NA	r7-70	NA	r5-60	**691.84**	r7-70	**777.86**
r6-48	NA	r7-84	NA	r6-48	**515.01**	r7-84	**906.67**
r6-60	NA	r8-64	NA	r6-60	**697.67**	r8-64	**654.98**
r6-72	NA	r8-80	NA	r6-72	**780.72**	r8-80	**814.64**
r7-56	NA	r8-96	NA	r7-56	**617.67**	r8-96	**1073.05**
Previous best solutions $\gamma = 0.4$				Current best solutions $\gamma = 0.4$			
Instance	BC(min)	Instance	BC(min)	Instance	BC(min)	Instance	BC(min)
c3-24	274.81	u3-24	67.56	c3-24	**274.80**	u3-24	67.56
r5-60	NA	r7-70	NA	r5-60	**719.68**	r7-70	**782.84**
r6-48	NA	r7-84	NA	r6-48	**509.71**	r7-84	**1001.10**
r6-60	NA	r8-64	NA	r6-60	**703.57**	r8-64	**646.15**
r6-72	NA	r8-80	NA	r6-72	**836.96**	r8-80	**833.13**
r7-56	NA	r8-96	NA	r7-56	**618.46**	r8-96	NA
Previous best solutions $\gamma = 0.7$				Current best solutions $\gamma = 0.7$			
Instance	BC(min)	Instance	BC(min)	Instance	BC(min)	Instance	BC(min)
c4-32	430.07	u4-32	99.50	c4-32	**406.94**	u4-32	99.50
c5-40	447.63	u5-40	NA	c5-40	**431.33**	u5-40	**128.86**
c5-50	NA	u5-50	144.36	c5-50	**622.56**	u5-50	144.36

lower that the 0.01% tolerance gap of Gurobi on instance c3-24. For Uber and Cordeau instances with parameter $\gamma = 0.7$, DA significantly improved the reported solutions for instances c4-32 and c5-40, with improvement gaps of respectively $5,36\%$ and $3,64\%$, and found a first feasible solution for instance c5-50 and u5-40. For some Uber and Cordeau instances with parameter $\gamma = 0.7$, we do not have a still feasible solution found, one may conjecture that many of these instances are infeasible for $\gamma = 0.7$, it is a perspective to prove it using exact methods or arguments with lower bounds.

These previous results illustrates the limit of solving capabilities of the exact B&C, even with a time limit of three hours, to solve E-DARP instances with 50 requests and more, especially for highly constrained instances. Our DA algorithm can tackle larger size of instances. Røpke instances for $\gamma = 0.1$ and $\gamma = 0.4$ are feasible, these results are the first solution found for these new instances, as a benchmark for future studies. We found no feasible solution for Røpke instances with $\gamma = 0.7$, despite 30 runs and 10000 iterations, one may conjecture also that many of these instances are to constrained to be feasible for $\gamma = 0.7$, it is a perspective to prove it using exact methods or arguments with lower bounds.

Detailed results for Røpke instances are gathered in Appendix A. The dispersion of results is appreciated calculating gaps for different runs to the new best known solution AC. We have similar results for the performance of DA, statistical dispersion increases from $\gamma = 0.1$ to $\gamma = 0.4$, but the dispersion remains quite acceptable. For instances r6-72, r7-84, most of the runs with $\gamma = 0.4$ do not find a feasible solution, but feasible solutions are found by siome runs, these instances seems challenging for future works.

To conclude, the proposed algorithm remains highly effective and can provide optimal/near-optimal solutions even facing highly constrained instances, it significantly outperforms the former state-of-the-art B&Cut algorithm for large instances, and the consistency of the algorithm seems quite acceptable for such difficult instances.

5.4. Allowing Multiple Visits to a Recharging Station

As discussed in section 3.5, the hypothesis to visit each recharging station at most once is not realistic. DA algorithm was designed to be generic to relax this constraints, and it allows to investigate the impact on solution cost and feasibility issues that are due to this non-realistic constraint.

The experiments on the three datasets with $\gamma \in \{0.1, 0.4, 0.7\}$ were also re-run allowing multiple visits to recharging stationss, the detailed results are in Appendix A. For such experiments, the gain in the cost is evaluated with DA with 10000 iterations and 30 runs in the standard version of E-ADARP as in [6] and with the multiple visits version.

With $\gamma = 0.1$, all the instances can be solved optimally/near-optimally without visiting recharging stations several times. Results obtained for both cases are similar, with a slight decrease in solution cost on Uber instances when allowing multiple visits. Allowing multiple visits cannot improve the results in this case 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. With $\gamma = 0.4$, it can be observed that allowing multiple visits to recharging stations can decrease the solution cost, and solve previously unsolved instance (i.e., u4-48). By carefully analyzing the routing result of each instance, 17.9% instances visit a recharging station multiple times. With $\gamma = 0.7$, allowing multiple visits to recharging stations can efficiently decrease the total cost. Moreover, all the unsolved instances in Cordeau/Uber datasets can be solved when the visits to recharging stations are not limited. Uber instances are easier to be solved than Cordeau instances because the number of recharging stations associated in each instance is larger than Cordeau instances. Cordeau instance set thus shows larger improvement compared to Uber instances while relaxing the visits to the recharging station. In case of high-energy-level constraint (i.e., $\gamma = 0.7$), 27 out of 28 instances have been further solved/improved while allowing multiple visits to recharging stations.

Røpke instances are all feasible with $\gamma = 0.4$ and $\gamma = 0.7$, and we do not have the previous difficulty to have instances where most of the runs do not find a feasible solution whereas one is found by some threads.

Generally, the previous difficulties for DA local search are lessened considering multiple visits to recharging stations. This induces that the difficulty of E-ADARP instances previously mentioned seems to be somewhat artificial, it is interesting to challenge local search approaches on highly-constrained instances, but more realistic constraints on capacity of recharging stations are helpful for the efficiency of the DA local search.

6. Conclusions and perspectives

This paper proposed a DA meta-heuristic for solving medium to large-sized E-ADARP instances, with problem-tailored neighborhoods and repairing operators. Seven types of neighborhoods are provided, three intra-route and four iter-route neighborhoods. To compute costs of neighboring solutions, we used an adapted version of the eight-step evaluation scheme from [12]. To repair infeasibility, 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 the proposed algorithm with comparisons to the former best-known results from [6]. The average gap between the best-obtained results and the former best-known results is 0.22%, we have found new best solutions on previously solved and unsolved instances, and on new benchmark instances with up to 8 vehicles and 96 requests. The statistical analysis of results for 30 independent runs shows a good accuracy of the DA local search.

A separate analysis of the results with three different energy threshold values $\gamma \in \{0.1; 0.4; 0.7\}$ allowed to examine the algorithm performance on loosely to highly constrained instances. Results illustrate that loosely constrained instances with $\gamma = 0.1$ are easier to solve for our DA local search with an excellent solution quality and a small dispersion of the results over the 30 independent runs. The corresponding instances with $\gamma = 0.4$ are still solvable by the DA algorithm, with a larger dispersion of the results. The corresponding instances with $\gamma = 0.7$ may be infeasible for many runs of the DA algorithm, some feasible solutions found by DA are the first feasible solutions found. Some instances remain unsolved after 30 independent runs of DA, one may conjecture no feasible solution exist for many of these instances, it is a perspective to prove it using exact methods. E-ADARP is an interesting case study to challenge a local search heuristic for highly constrained instances, with γ varying.

To explain having so highly constrained instances, we consider a more realistic situation in which multiple visits to recharging stations are allowed. The previous difficulties for DA local search are lessened considering this more realistic situation, feasible solutions are consistently found and the dispersion of results is smaller. This induces that the difficulty of E-ADARP instances previously mentioned seems to be somewhat artificial, it is interesting to challenge local search approaches on highly-constrained instances, but more realistic constraints on capacity of recharging stations are helpful for the efficiency of the DA algorithm. It opens perspective in the way of modeling constraints for recharging stations, for instance considering capacity and scheduling constraints in the use of recharging stations.

The weakness of the proposed DA algorithm is that the consistency decreases for highly-constrained instances. It is coherent with the use of local search and meta-heuristics for such difficult instances. It is a perspective to try to improve these results. First note that the proposed DA algorithm was designed and evaluated in a sequential mode. A parallel version for the DA algorithm is likely to improve the consistency of the local search in the same computation times, as in [30]. Similarly, a population-based meta-heuristic like an Evolutionay Algorithm (EA) is another way to increase the result steadiness with more diversification and also parallelization [31, 32] . Our local search operators may be re-used inside EA to provide mutation and reparation operators. Another standard way of dealing with highly constrained instances is to use mathematical programming to explore large neighborhoods with many infeasibilities, as in [7]. Adding matheuristic operators, a trade-off has to be found between the small

neighborhoods with quick exploration capabilities and the larger neighborhoods requiring a larger computation time for an efficient exploration. This offers some perspectives to try to improve the results of the DA algorithm for highly-constrained instances.

Another perspective would be to challenge and adapt our DA local search on much larger instances. The maximal size of instances was 96 requests and 8 vehicles based on existing instances from the literature for DARPs. Contrary to the B&C algorithm which becomes inefficient with more than 50 requests, our DA local search may be used for larger instances.

Furthermore, 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' inconvenience 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 these additional constraints. Having several objective functions, a perspective is to design population-based multi-objective meta-heuristics, like EAs, to handle conflicting objectives. 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 crucial in such a context where meta-heuristics seem also promising.

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] N. H. Wilson, J. M. Sussman, H.-K. Wong, T. Higonnet, Scheduling algorithms for a dial-a-ride system, Massachusetts Institute of Technology. Urban Systems Laboratory, 1971.

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

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

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

[7] N. Dupin, R. Parize, E.-G. Talbi, Matheuristics and column generation for a basic technician routing problem, Algorithms 14 (11) (2021) 313.

[8] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, A. A. Juan, Rich vehicle routing problem: Survey, ACM Computing Surveys (CSUR) 47 (2) (2014) 1–28.

[9] V. Schmid, K. F. Doerner, G. Laporte, Rich routing problems arising in supply chain management, European Journal of Operational Research 224 (3) (2013) 435–448.

[10] T. Vidal, T. G. Crainic, M. Gendreau, C. Prins, A unified solution framework for multi-attribute vehicle routing problems, European Journal of Operational Research 234 (3) (2014) 658–673.

[11] P. Penna, A. Subramanian, L. S. Ochi, T. Vidal, C. Prins, A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet, Annals of Operations Research (2017) 1–70.

[12] J.-F. Cordeau, G. Laporte, A tabu search heuristic for the static multi-vehicle dial-a-ride problem, Transportation Research Part B: Methodological 37 (6) (2003) 579–594.

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

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

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

[16] 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.

[17] G. Desaulniers, F. Errico, S. Irnich, M. Schneider, Exact algorithms for electric vehicle-routing problems with time windows, Operations Research 64 (6) (2016) 1388–1405.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

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

[27] 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.

[28] 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.

[29] M. W. Savelsbergh, Preprocessing and probing techniques for mixed integer programming problems, ORSA Journal on Computing 6 (4) (1994) 445–454.

[30] N. Dupin, E. Talbi, Parallel matheuristics for the discrete unit commitment problem with min-stop ramping constraints, International Transactions in Operational Research 27 (1) (2020) 219–244.

[31] 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.

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

Appendix A: Intermediate result Tables

Tables 8, 9, 10 and 11 compare the results of proposed DA algorithm with BKS from [6] for Cordeau and Uber instances considering single visits of recharging stations, with parameters $\gamma \in \{0.1, 0.4\}$. Aggregated results were presented in Table 6.

Tables 12 and 13 present similar tables with Tables 8, 9, 10 and 11, with parameter $\gamma = 0.7$, feasible solutions are available only in subsets. New Best solutions were reported in Table 7.

Table 14 present the results of proposed DA algorithm for Røpke instances considering single visits of recharging stations, with parameters $\gamma \in \{0.1, 0.4\}$, considering BC as BKS to appreciate the dispersion gaps.

Tables 15, 16, 17 and 18 present the corresponding results allowing multiple visits per recharging station.

Table 8

DA results (30 runs, 10000 iterations per run) on Cordeau instances with $\gamma = 0.1$

Instance	DA algorithm						BKS[6][a]	
$\gamma = 0.1$	BC(min)	BC%	AC%	Q1%	Q3%	AT(s)	Obj(min)	CPU(s)
c2-16	**237.38**	0	0	0	0	355.84	237.38*	1.2
c2-20	**279.08**	0	0	0	0	768.99	279.08*	4.2
c2-24	**346.21**	0	0.10%	0	0	1086.87	346.21*	9.0
c3-18	**236.82**	0	0	0	0	214.53	236.82*	4.8
c3-24	**274.81**	0	0.28%	0	0.74%	411.87	274.81*	13.80
c3-30	**413.27**	0	0.11%	0	0	761.52	413.27*	102
c3-36	481.40	0.05%	0.25%	0.05%	0.25%	1102.75	481.17*	106.80
c4-16	**222.49**	0	0.04%	0	0	114.1	222.49*	3.6
c4-24	**310.84**	0	0.37%	0	0.70%	245.79	310.84*	31.2
c4-32	**393.96**	0	1.30%	0.77%	1.81%	446.93	393.96*	612
c4-40	**453.84**	0	1.29%	0.97%	1.57%	802.28	453.84*	517.2
c4-48	558.62	0.74%	2.01%	1.55%	2.56%	1355.42	554.54	7200
c5-40	414.80	0.07%	2.22%	1.75%	2.57%	501.59	414.51*	1141.8
c5-50	565.46	1.12%	3.07%	2.42%	3.61%	871.19	559.17	7200
Avg		**0.14%**	**0.79%**	**0.54%**	**0.99%**	645.69		1210.54

[a] Two-index model's results from [6] running on a 3.6 GHz Intel(R) Core(TM) with 16 Gb of RAM.

* Numbers with star indicate it is the optimal value proved in [6].

Table 9

DA results (30 runs, 10000 iterations per run) on Uber instances with $\gamma = 0.1$

Instance	DA algorithm						BKS[6][a]	
$\gamma = 0.1$	BC(min)	BC%	AC%	Q1%	Q3%	AT(s)	Obj(min)	CPU(s)
u2-16	**57.61**	0	0	0	0	478.79	57.61*	21
u2-20	**55.59**	0	0	0	0	837.69	55.59*	9.6
u2-24	**91.27**	0	0.37%	0	0	1307.44	91.27*	432
u3-18	**50.74**	0	0.13%	0	0.37%	257.79	50.74*	10.8
u3-24	**67.56**	0	0.51%	0.44%	0.59%	497.88	67.56*	130.2
u3-30	**76.75**	0	0.65%	0	1.04%	883.2	76.75*	438
u3-36	104.13	0.08%	1.48%	0.47%	2.02%	1547.4	104.04*	1084.8
u4-16	**53.58**	0	0.04%	0	0	139.69	53.58*	48
u4-24	89.96	0.14%	1.30%	1.15%	1.57%	240.25	89.83*	13.2
u4-32	**99.29**	0	0.50%	0.09%	0.81%	501.82	99.29*	1158.6
u4-40	**133.11**	0	2.02%	1.01%	3.10%	807.85	133.11*	185.4
u4-48	148.59	0.20%	1.52%	0.54%	2.62%	535.09	148.30	7200
u5-40	121.95	0.08%	1.72%	1.02%	2.75%	1049.92	121.86	7200
u5-50	144.03	0.65%	2.25%	1.50%	2.69%	478.79	143.10	7200
Avg		**0.08%**	**0.89%**	**0.44%**	**1.25%**	738.04		1280.83

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

* Numbers with star indicate it is the optimal value proved in [6].

Table 10

DA results (30 runs, 10000 iterations per run) on Cordeau instances with $\gamma = 0.4$

Instance	DA algorithm						BKS[6][a]	
$\gamma = 0.4$	BC(min)	BC%	AC%	Q1%	Q3%	AT(s)	Obj(min)	CPU(s)
A2-16	**237.38**	0	0	0	0	387.79	237.38*	1.8
A2-20	**280.70**	0	0.28%	0	0	761.59	280.70*	49.8
A2-24	350.68	0.76%	0.82%	0.76%	0.84%	1106.41	348.04*	25.2
A3-18	**236.82**	0	0	0	0	214.83	236.82*	4.2
A3-24	**274.80***	0	0.44%	0	0.74%	439.29	274.81*	16.8
A3-30	**413.37**	0	0.004%	0	0	818.59	413.37*	99
A3-36	485.37	0.25%	1.23%	0.68%	1.42%	1105.85	484.14*	306.6
A4-16	**222.49**	0	0.06%	0	0	114.45	222.49*	5.4
A4-24	**311.03**	0	0.46%	0.20%	0.70%	237.75	311.03*	39.6
A4-32	**394.26**	0	1.15%	0.47%	1.61%	458.7	394.26*	681.6
A4-40	**453.84**	0	1.66%	1.16%	2.37%	819.41	453.84*	417.6
A4-48	565.38	1.94%	3.67%	3.04%	4.46%	1375.89	554.60	7200
A5-40	416.87	0.57%	1.95%	1.58%	2.42%	481.98	414.51*	1221
A5-50	572.77	2.19%	4.06%	3.29%	4.82%	826.3	560.50	7200
Avg		**0.41%**	**1.13%**	**0.80%**	**1.38%**	653.49		1233.47

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

* Numbers with star indicate it is the optimal value proved in [6].

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

Table 11

DA results (30 runs, 10000 iterations per run) on Uber instances with $\gamma = 0.4$

Instance	DA algorithm						BKS[6][a]	
$\gamma = 0.4$	BC(min)	BC%	AC%	Q1%	Q3%	AT(s)	Obj(min)	CPU(s)
u2-16	**57.65**	0	0	0	0	506.57	57.65*	25.8
u2-20	**56.34**	0	0	0	0	953.1	56.34*	12
u2-24	**91.63**	0	0	0	0	1232.86	91.63*	757.2
u3-18	**50.74**	0	0.15%	0	0.50%	288.53	50.74*	13.8
u3-24	67.67	0.16%	0.60%	0.46%	0.82%	495.88	67.56*	220.8
u3-30	**76.75**	0	1.59%	0.98%	2.76%	967.65	76.75*	336.6
u3-36	104.90	0.81%	2.83%	2.27%	3.36%	1657.75	104.06*	2010
u4-16	**53.58**	0	0.03%	0	0	152.84	53.58*	44.4
u4-24	90.15	0.35%	1.42%	1.20%	1.64%	265.75	89.83*	28.2
u4-32	**99.29**	0	0.68%	0.05%	0.89%	527.25	99.29*	2667.6
u4-40	134.01	0.08%	2.44%	0.97%	3.81%	877.36	133.91*	2653.2
u4-48	NA	NA	NA	NA	NA	1762.75	NA	7200
u5-40	**122.23**	0	1.61%	0.62%	2.44%	560.8	122.23	7200
u5-50	145.36	1.55%	2.78%	2.35%	3.24%	1001.93	143.14	7200
Avg		**0.23%**	**1.09%**	**0.68%**	**1.50%**	803.64		2169.25

[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 with star indicate it is the optimal value proved in [6].

Table 12

DA results (30 runs, 10000 iterations per run) on Cordeau instances with $\gamma = 0.7$

Instance	DA algorithm						BKS[6][a]	
$\gamma = 0.7$	BC(min)	BC%	AC%	Q1%	Q3%	AT(s)	Obj(min)	CPU(s)
c2-16	**240.66**	0	0	0	0	467.89	240.66*	5.4
c2-20	NA	NA	NA	NA	NA	816.99	NA	7200
c2-24	**358.21**	0	NA	1.45%	3.60%	1149.69	358.21*	961.2
c3-18	**240.58**	0	0single 0.7 uber	0	0	284.30	240.58*	48
c3-24	280.97	1.17%	2.09%	1.59%	2.29%	472.27	277.72*	152.4
c3-30	NA	NA	NA	NA	NA	806.19	NA	7200
c3-36	501.60	1.53%	NA	1.93%	3.49%	1196.24	494.04	7200
c4-16	**223.13**	0	0	0	0	137.51	223.13*	67.2
c4-24	318.97	0.24%	0.87%	0.26%	1.46%	281.92	318.21*	1834.8
c4-32	**406.94***	-5.38%*	NA	-0.42%	NA	501.99	430.07	7200
c4-40	NA	NA	NA	NA	NA	935.97	NA	7200
c4-48	NA	NA	NA	NA	NA	1486.16	NA	7200
c5-40	**431.33***	-3.64%*	-0.71%	-2.14%	0.72%	546.92	447.63	7200
c5-50	**622.56***	NA	NA	NA	NA	924.79	NA	7200
Avg		NA[b]	NA[b]	NA[b]	NA[b]	714.92		4333.5

[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 with star indicate it is the optimal value proved in [6].

* Numbers in bold with star indicate new best solution found by DA, the negative values mean the improvement on the objective function.

Table 13

DA results (30 runs, 10000 iterations per run) on Uber instances with $\gamma = 0.7$

Instance	DA algorithm						BKS[6][a]	
$\gamma = 0.7$	BC(min)	BC%	AC%	Q1%	Q3%	AT(s)	Obj(min)	CPU(s)
u2-16	**59.19**	0	0	0	0	458.12	59.19*	338.4
u2-20	**56.86**	0	3.09%	2.29%	3.12%	1057.23	56.86*	72
u2-24	NA	NA	NA	NA	NA	274.48	NA	7200
u3-18	**50.99**	0	1.14%	0	1.32%	481.98	50.99*	24
u3-24	68.39	0	0.75%	0.45%	1.00%	997.27	68.39*	400.2
u3-30	78.28	0.18%	1.84%	0.76%	2.54%	1003.24	78.14*	3401.4
u3-36	107.13	1.27%	NA	NA	NA	124.61	105.79	7200
u4-16	**53.87**	0	5.73%	5.40%	5.67%	260.32	53.87*	88.8
u4-24	90.97	1.12%	2.42%	1.89%	2.72%	344.52	89.96*	22.8
u4-32	103.12	3.63%	NA	NA	NA	716.4	99.50*	2827.2
u4-40	NA	NA	NA	NA	NA	869.18	NA	7200
u4-48	NA	NA	NA	NA	NA	446.0	NA	7200
u5-40	**128.86***	NA	NA	NA	NA	744.74	NA	7200
u5-50	144.75	0.27%	6.50%	0.55%	5.96%	458.12	144.36	7200
Avg		NA[b]	NA[b]	NA[b]	NA[b]	598.32		4333.5

[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 with star indicate it is the optimal value proved in [6].

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

Table 14

DA results (30 runs, 1000 iterations per run) on Røpke instances with recharging station being visited at most once

$\gamma = 0.1$	BC(min)	AC%	Q1%	Median%	Q3%	AT(s)
r5-60	691.84	2.73%	1.73%	2.46%	3.85%	1314.30
r6-48	515.01	1.86%	0.90%	1.74%	2.45%	567.88
r6-60	697.67	2.00%	1.24%	1.82%	2.37%	875.03
r6-72	780.72	2.32%	1.37%	2.44%	2.86%	1446.90
r7-56	617.67	2.54%	1.67%	2.64%	3.61%	538.80
r7-70	777.86	4.18%	3.06%	4.09%	5.49%	926.47
r7-84	906.67	2.27%	0.71%	2.21%	3.46%	1507.28
r8-64	654.98	2.48%	1.44%	2.41%	3.41%	612.44
r8-80	814.64	1.83%	0.95%	1.59%	2.46%	982.31
r8-96	1073.05	2.12%	1.20%	1.99%	2.81%	1580.94
$\gamma = 0.4$	BC(min)	AC%	Q1%	Median%	Q3%	AT(s)
r5-60	719.68	2.90%	1.39%	3.34%	4.25%	1861.60
r6-48	509.71	2.41%	1.76%	2.42%	3.18%	709.77
r6-60	703.57	2.77%	1.62%	2.81%	3.74%	1157.06
r6-72	836.96	NA	5.25%	NA	NA	2412.51
r7-56	618.46	2.68%	1.75%	2.85%	3.59%	693.68
r7-70	782.84	4.05%	1.92%	4.15%	5.69%	1207.86
r7-84	1001.10	NA	NA	NA	NA	2090.81
r8-64	646.15	2.71%	1.70%	2.68%	3.77%	764.26
r8-80	833.13	2.61%	1.43%	2.59%	3.72%	1415.64

Table 15

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
c2-16	237.38	237.38	0	u2-16	57.61	57.61	0
c2-20	279.08	279.08	0	u2-20	55.59	55.59	0
c2-24	346.21	346.21	0	u2-24	**90.54****	91.27	-0.81%
c3-18	236.82	236.82	0	u3-18	50.74	50.74	0
c3-24	274.80	274.80	0	u3-24	67.56	67.56	0
c3-30	413.27	413.27	0	u3-30	76.75	76.75	0
c3-36	481.40	481.40	0	u3-36	**104.08****	104.13	-0.05%
c4-16	222.49	222.49	0	u4-16	53.58	53.58	0
c4-24	310.84	310.84	0	u4-24	**89.83****	89.96	-0.14%
c4-32	393.96	393.96	0	u4-32	99.29	99.29	0
c4-40	455.08	453.84	0.27%	u4-40	134.12	133.11	0.75%
c4-48	560.29	558.62	0.30%	u4-48	**147.94****	148.59	-0.44%
c5-40	419.04	414.80	1.01%	u5-40	122.23	121.95	0.23%
c5-50	568.35	565.46	0.51%	u5-50	144.03	144.03	0
Avg			0.15%				-0.03%

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

Table 16

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
c2-16	237.38	237.38	0	u2-16	57.65	57.65	0
c2-20	280.70	280.70	0	u2-20	56.34	56.34	0
c2-24	**348.92****	350.68	-0.50%	u2-24	**90.28****	91.63	-1.50%
c3-18	236.82	236.82	0	u3-18	50.74	50.74	0
c3-24	274.80	274.80	0	u3-24	**67.56****	67.67	-0.16%
c3-30	413.37	413.37	0	u3-30	76.75	76.75	0
c3-36	**481.40****	485.37	-0.82%	u3-36	**103.86****	104.90	-1.00%
c4-16	222.49	222.49	0	u4-16	53.58	53.58	0
c4-24	311.03	311.03	0	u4-24	90.30	90.15	0.17%
c4-32	395.32	394.26	0.27%	u4-32	99.29	99.29	0
c4-40	453.84	453.84	0	u4-40	**133.65****	134.01	-0.27%
c4-48	**556.94****	565.38	-1.52%	u4-48	**148.84****	NA	NA
c5-40	**416.80****	416.87	-0.02%	u5-40	122.31	122.23	0.07%
c5-50	**568.58****	572.77	-0.74%	u5-50	**144.59****	145.36	-0.53%
Avg			-0.24%				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.

Table 17

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
c2-16	240.66	240.66	0	u2-16	**57.65****	59.19	-2.67%
c2-20	**282.32****	NA	NA	u2-20	**56.34****	56.86	-0.92%
c2-24	**350.39****	358.21	-2.23%	u2-24	**90.28****	NA	NA
c3-18	**240.04****	240.58	-0.22%	u3-18	**50.74****	50.99	-0.49%
c3-24	**276.27****	280.97	-1.70%	u3-24	**67.56****	68.39	-1.23%
c3-30	**418.67****	NA	NA	u3-30	**76.75****	78.28	-1.99%
c3-36	**493.58****	501.60	-1.62%	u3-36	**103.86****	107.13	-3.15%
c4-16	**222.49****	223.13	-0.29%	u4-16	**53.58****	53.87	-0.54%
c4-24	**316.79****	318.97	-0.69%	u4-24	**90.30****	90.97	-0.74%
c4-32	**395.79****	406.94	-2.82%	u4-32	**99.29****	103.12	-3.86%
c4-40	**466.68****	NA	NA	u4-40	**133.65****	NA	NA
c4-48	**574.69****	NA	NA	u4-48	**148.84****	NA	NA
c5-40	**417.98****	431.33	-3.19%	u5-40	**122.30****	128.86	-5.36%
c5-50	**583.95****	622.56	-6.61%	u5-50	**144.59****	144.75	-0.11%
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.

Table 18

DA results (30 runs, 1000 iterations per run) on Røpke instances with recharging station being visited multiple times ($\gamma = 0.1, 0.4, 0.7$)

$\gamma = 0.1$	BC(min)	AC%	Q1%	Median%	Q3%	AT(s)
r5-60	693.05	2.73%	1.73%	2.46%	3.85%	1321.17
r6-48	510.70	1.86%	0.90%	1.74%	2.45%	564.95
r6-60	695.00	2.00%	1.24%	1.82%	2.37%	886.86
r6-72	790.54	2.32%	1.37%	2.44%	2.86%	1421.36
r7-56	619.48	2.54%	1.67%	2.64%	3.61%	533.06
r7-70	765.84	4.18%	3.06%	4.09%	5.49%	931.26
r7-84	907.50	2.27%	0.71%	2.21%	3.46%	1497.72
r8-64	647.80	2.48%	1.44%	2.41%	3.41%	618.75
r8-80	830.80	1.83%	0.95%	1.59%	2.46%	1013.93
r8-96	1078.55	2.12%	1.20%	1.99%	2.81%	1541.38
$\gamma = 0.4$	BC(min)	AC%	Q1%	Median%	Q3%	AT(s)
r5-60	698.51	2.65%	2.13%	2.64%	3.27%	1853.02
r6-48	513.86	1.50%	0.76%	1.52%	2.40%	717.66
r6-60	699.08	1.58%	0.99%	1.73%	2.20%	1080.19
r6-72	795.02	2.22%	1.27%	2.11%	3.02%	2036.00
r7-56	616.37	2.49%	1.74%	2.71%	3.27%	663.99
r7-70	774.34	2.91%	1.35%	2.98%	4.14%	1138.92
r7-84	901.58	2.96%	1.83%	2.71%	4.22%	1903.22
r8-64	649.33	2.04%	1.07%	1.93%	2.65%	724.87
r8-80	829.50	2.28%	1.64%	2.33%	2.93%	1323.61
r8-96	1075.70	3.90%	2.92%	3.78%	5.51%	2334.14
$\gamma = 0.7$	BC(min)	AC%	Q1%	Median%	Q3%	AT(s)
r5-60	703.06	3.10%	2.07%	3.19%	4.13%	3895.01
r6-48	516.22	2.27%	1.12%	2.15%	3.50%	1231.48
r6-60	701.02	1.83%	0.93%	1.67%	2.60%	1930.83
r6-72	812.28	2.56%	1.65%	2.76%	3.55%	4449.72
r7-56	630.65	3.34%	2.46%	3.29%	4.37%	1394.16
r7-70	783.56	3.95%	2.98%	3.97%	4.81%	2431.33
r7-84	915.43	3.06%	1.81%	3.35%	3.97%	3301.70
r8-64	658.82	2.23%	1.43%	1.89%	2.60%	1355.50
r8-80	854.29	2.10%	0.91%	1.90%	2.80%	2793.81
r8-96	1087.57	4.43%	3.90%	4.69%	5.22%	4359.38