A RIDE TIME-ORIENTED SCHEDULING ALGORITHM FOR DIAL-A-RIDE PROBLEMS

Claudia Bongiovanni  
Polytechnique Montréal and CIRRELT  
claudia.bongiovanni@polymtl.ca

Nikolas Geroliminis  
École Polytechnique Fédérale de Lausanne  
nikolas.geroliminis@epfl.ch

Mor Kaspi  
Tel-Aviv University  
morkaspi@tauex.tau.ac.il

ABSTRACT

This paper offers a new algorithm to efficiently optimize scheduling decisions for dial-a-ride problems (DARPs), including problem variants considering electric and autonomous vehicles (e-ADARPs). The scheduling heuristic, based on linear programming theory, aims at finding minimal user ride time schedules in polynomial time. The algorithm can either return optimal feasible routes or it can return incorrect infeasibility declarations, on which feasibility can be recovered through a specifically-designed heuristic. The algorithm is furthermore supplemented by a battery management algorithm that can be used to determine charging decisions for electric and autonomous vehicle fleets. Timing solutions from the proposed scheduling algorithm are obtained on millions of routes extracted from DARP and e-ADARP benchmark instances. They are compared to those obtained from a linear program, as well as to popular scheduling procedures from the DARP literature. Results show that the proposed procedure outperforms state-of-the-art scheduling algorithms, both in terms of compute-efficiency and solution quality.

Keywords scheduling · routing · battery management · dial-a-ride · heuristics

1 Introduction

Many real-world urban mobility and supply chain problems involve time-dependent allocation and sequencing decisions that must be frequently re-optimized over time. The re-optimization process is typically composed of the production and evaluation of a plethora of potential solutions. The evaluation of these solutions is performed by scheduling algorithms, whose goals are to determine the timing of the sequential decisions, while respecting several time-related constraints and objectives (e.g., Vidal et al. 2015). Their efficient resolution is particularly relevant for large-scale and dynamic problems, since in these cases scheduling decisions must be made frequently and quickly, as for example in the context of large neighborhood search methods (e.g., Gschwind and Drexel 2019). In addition to computational efficiency, it is essential that scheduling algorithms provide high-quality solutions, given their direct impact on the quality of allocation decisions and thus on the overall performance of the optimization strategy.

Scheduling algorithms are of utmost importance for hardly-constrained vehicle routing problems. They assume feasible vehicle routes in terms of capacity and precedence constraints and are designed to provide feasible decisions on time-related variables (e.g., service start times and waiting times at each node). Demonstrating the feasibility of a fixed route and composing high-quality vehicle schedules is a challenging task for most vehicle routing problems because of the presence of time-related constraints (e.g., time windows and maximum route duration). For dial-a-ride (DARP) problems, this task is made even more challenging by the presence of constraints on maximum user ride time (Cordeau and Laporte 2007), as well as constraints on battery management and charging time for fleets composed of electric and autonomous vehicles (e-ADARP, Bongiovanni et al. 2019, Yue et al. 2022, 2023). Consequently, the goal of some scheduling algorithms for DARPs is optimize time dependent level of service while satisfying several time related...
constraints. The former is often measured by the user excess ride time, that is the delay users experience for sharing rides as compared to a taxi service (e.g., see Section 3 in Molenbruch et al. 2017a).

A variety of heuristics have been proposed in the DARP literature to manage path planning. However, these procedures typically face a trade-off between returning correct feasibility declarations and increasing the quality of the returned solutions. Cordeau and Laporte (2003) propose an 8-step procedure setting the earliest start time at each vertex in the route and using forward slack times to delay the start times at pickup locations in vision of maximum ride time constraints (Savelsbergh 1992). Parragh et al. (2009) observe that adopting a sequential approach to avoid maximum ride time violations does not necessarily minimize the total user excess ride time in the schedules. In fact, delaying the service start time at a given pickup location may decrease the excess ride time of the specific request but increase the excess ride time of other requests in the route. As such, Parragh et al. (2009) modify the procedure in Cordeau and Laporte (2003) by adapting the computation of forward slack times such that increases in the user excess ride time of any request in the route is avoided. As a result, the returned feasible schedules minimize the total user excess ride time at the expense of incorrect infeasibility declarations. This last aspect is a drawback that is tackled in Molenbruch et al. (2017b), who propose a procedure starting by considering a possibly travel-time infeasible schedule setting the excess ride time of each user at its lowest bound. Infeasibility relates to travel time shortages between successive nodes and is successively recovered by shifting service start times such that the total user excess ride time is minimized.

In this work, we propose a novel scheduling heuristic which is numerically shown to provide excess-time optimal solutions in most cases, although it may result with incorrect infeasibility declarations or suboptimal solutions. The designed algorithm is complemented by a battery management heuristic to deal with DARP extensions employing electric autonomous vehicles, i.e., e-ADARPs. The goal of the battery management heuristic is to assign charging times at the visited stations. Indeed, a schedule minimizing excess ride time may not be battery feasible, and vice versa. The procedure builds on two main observations: (i) minimizing excess-time while respecting time-window and user-ride-time constraints is in fact equivalent to assigning the right amount of waiting time at all nodes in the routes; and (ii) ensuring battery feasibility is in fact possible by recharging as much as possible at the visited facilities, as early as possible. The proposed scheduling and battery management algorithms are tested on millions of DARP and e-ADARP routing solutions obtained by developing an adaptive large neighborhood search heuristic (Ropke and Pisinger 2006). The quality of the obtained scheduling solutions are compared against: (i) a linear program; (ii) the well-known 8-step scheduling procedures by Cordeau and Laporte (2003) and Parragh et al. (2009); and (iii) the procedure by Molenbruch et al. (2017b).

The rest of the paper is organized as follows: Section 2 introduces the excess-ride-time scheduling problem, Section 3 provides the scheduling algorithm, Section 4 presents the battery management problem and the heuristic procedure to provide battery-feasibility. Finally, Section 5 provides numerical experiments comparing the scheduling algorithm against state-of-the-art procedures, and Section 6 summarizes the main concepts of this paper and provides an overlook to future research.

## 2 Scheduling Problem

Consider a predetermined route sequence \( I \) of \( M \) nodes, which includes pickup locations, dropoff locations, and potentially some charging stations. Without loss of generality, assume that the sequence satisfies routing constraints, as well as precedence and load constraints. Note that the given sequence may not necessarily satisfy all timing and battery management constraints. Then, the optimization problem consists in scheduling the service start times in the sequence as to minimize the total user excess ride time, guarantee battery, time window, and maximum ride time feasibility.

Start by considering a specific sub-sequence \( I \) of \( M \leq M \) nodes, which is one of the derived sub-sequences obtained by splitting \( I \) by the visited charging stations. Without loss of generality, assume that the visited charging station at the beginning of \( I \) represents an origin depot and the charging station at the end of \( I \) represents a destination depot. Furthermore, denote by \( i \in \{1, \ldots, n\} \) the set of requests contained in sequence \( I \), \( P_i \) the set of pickups, and \( D_i \) the set of dropoffs. Pickup locations are characterized by loads \( l_{P_i} \) and dropoff locations by loads \( l_{D_i} = -l_{P_i} \). Furthermore, all locations \( i, j \in I \) feature service times \( s_i \), direct travel times \( t_{i,j} \), and time windows \([\text{arr}_i, \text{dep}_i]\) limiting the time at which service may start. Note that, it is possible to set time windows around the pickup locations and derive the time windows around the corresponding dropoff locations by means of the maximum ride times \( u_{P_i} \), and vice versa (Cordeau and Laporte 2003). The goal is to be able to optimize \( I \) with respect to the total excess ride time by setting optimal service start times \( T_i \) with \( i \in \{1, \ldots, M\} \) in vision of maximum ride time and time window constraints. As
such, the scheduling problem for sub-sequence \( \bar{I} \) can be stated as the following linear program:

\[
(LP1) \quad \min \sum_{i \in \{1, \ldots, n\}} (T_{D_i} - T_{F_i} - d_{P_i} - t_{P_i,D_i})
\]

s.t.:

\[
T_i + t_{i,i+1} + s_i \leq T_{i+1} \quad \forall i \in \{1, 2, \ldots, \bar{M} - 1\}
\]

\[
T_{D_i} - T_{F_i} - d_{P_i} \leq w_{P_i} \quad \forall i \in \{1, \ldots, n\}
\]

\[
arr_i \leq T_i \leq dep_i \quad \forall i \in \{1, 2, \ldots, \bar{M}\}
\]

where the objective function (1) minimizes the sum of excess ride times of all requests in \( \bar{I} \), constraints (2) set the service start times between consecutive nodes, while constraints (3)-(4) impose maximum ride time and time window constraints respectively.

Note that time windows \([\text{arr}_i, \text{dep}_i]\) can be tightened in light of the travel times and service times between consecutive nodes. As such, by sequentially inspecting the sequence, it is possible to calculate the earliest time \( ET_i \) and latest time \( LT_i \) at which service can start at node \( i \) by using the following recursive formulas:

\[
ET_i = \max\{\text{arr}_1, ET_{i-1} + t_{i-1,i} + s_i\} \quad \forall i \in \{2, \ldots, \bar{M}\}, ET_1 = \text{arr}_1
\]

\[
LT_i = \max\{\text{dep}_{\bar{M}}, LT_{i+1} - t_{i,i+1} - s_i\} \quad \forall i \in \{1, \ldots, \bar{M} - 1\}, LT_{\bar{M}} = \text{dep}_{\bar{M}}
\]

Hence, a tighter formulation to LP1 can be obtained by substituting constraints (4) with:

\[
ET_i \leq T_i \leq LT_i \quad \forall i \in \{1, 2, \ldots, \bar{M}\}
\]

Service start time \( T_i \) at node \( i \) depends on the initial departure time from the depot, the total travel time up to \( i \), the total service time spent serving nodes before \( i \), and the total vehicle waiting time up to \( i \). That is:

\[
T_i = T_1 + \sum_{j=1}^{i-1} t_{j,j+1} + \sum_{j=1}^{i-1} d_j + \sum_{j=1}^{i} w_j
\]

Here, \( w_i \) denotes the waiting time at node \( i \). Note that, service may start as soon as possible without penalizing the objective function, i.e. \( T_1 = ET_1 \). With such representation of service start times, the objective function (1) can be re-written as follows:

\[
\min \sum_{i \in \{1, \ldots, n\}} (T_1 + \sum_{j=1}^{D_i-1} t_{j,j+1} + \sum_{j=1}^{D_i-1} d_j + \sum_{j=1}^{D_i-1} w_j) - (T_1 + \sum_{j=1}^{D_i-1} t_{j,j+1} + \sum_{j=1}^{D_i-1} d_j + \sum_{j=1}^{D_i-1} w_j) - d_{P_i} - t_{P_i,D_i} =
\]

\[
\min \sum_{i \in \{1, \ldots, n\}} \left( \sum_{j=P_i}^{D_i-1} t_{j,j+1} + \sum_{j=P_i}^{D_i-1} d_j + \sum_{j=P_i}^{D_i-1} w_j - d_{P_i} - t_{P_i,D_i} \right)
\]

Since travel times between consecutive nodes and the service times are deterministic parameters, minimizing the objective function (8) is equivalent to:

\[
\min \sum_{i \in \{1, \ldots, n\}} \sum_{j=P_i}^{D_i} w_j = \min \sum_{i=1}^{\bar{M}} L_i w_i
\]

Where \( L_i = \sum_{j=1}^{i-1} l_j \) represents the vehicle load up to node \( i \).

Equivalent to the re-writing of the objective function, constraints (7) can be re-defined through (8) as follows:

\[
\sum_{j=1}^{i} w_j \geq ET_i - \sum_{j=1}^{i-1} t_{j,j+1} + \sum_{j=1}^{i-1} d_j - ET_1 \quad \forall i \in \{1, 2, \ldots, \bar{M}\}
\]

\[
\sum_{j=1}^{i} w_j \leq LT_i - \sum_{j=1}^{i-1} t_{j,j+1} + \sum_{j=1}^{i-1} d_j - ET_1 \quad \forall i \in \{1, 2, \ldots, \bar{M}\}
\]

Note that these constraints provide lower and upper bounds to the total waiting time amount that needs to be distributed.
between \( i \) and all nodes preceding \( i \). As such, the linear program (LP1) can be equivalently re-defined as the following linear program:

\[
\text{(LP2)} \quad \min \sum_{i=1}^{M} L_i w_i
\]

s.t.:

\[
\sum_{j=1}^{i} w_j \geq ET_i - \sum_{j=1}^{i-1} t_{j,j+1} - \sum_{j=1}^{i-1} d_j - ET_1 \quad \forall i \in \{1, 2, \ldots, M\}
\]

\[
\sum_{j=1}^{i} w_j \leq LT_i - \sum_{j=1}^{i-1} t_{j,j+1} - \sum_{j=1}^{i-1} d_j - ET_1 \quad \forall i \in \{1, 2, \ldots, M\}
\]

\[
\sum_{j=i+1}^{M} w_j \leq u_i - \sum_{j=i}^{i-1} t_{j,j+1} - \sum_{j=i+1}^{M-1} d_j \quad \forall i \in P
\]

Remark that constraints (2) from (LP1) are guaranteed by the definition of equation (8), which now composes constraints (14) and (15). The right-hand side of (14) and (15) represent the minimal total waiting time that must be assigned up to node \( i \), and the maximal total waiting time that can be assigned to node \( i \), without violating the time windows at successive nodes in the sequence. For convenience, denote the right-hand side of constraints (14) and (15) by \( \Delta_i \) and \( \Theta_i \), respectively. Using equations (5) and (6) we have that \( \Delta_i \leq \Delta_{i+1} \quad \forall i \in \{1, 2, \ldots, M-1\} \) and \( \Theta_i \leq \Theta_{i+1} \quad \forall i \in \{1, 2, \ldots, M-1\} \). That is, the total minimal and maximal waiting time that needs to be distributed in the sequence may only increase between consecutive nodes. As such, note that the total minimal waiting time \( \Delta_M \), i.e. at the destination depot, represents a waiting time amount that cannot be avoided in any feasible solution. Finally, the objective of the problem reduces to optimally distribute \( \Delta_M \) among all nodes \( \{1, \ldots, M\} \) in consideration of the total load \( L_i \) at each node in the sequence, which impacts the total user excess ride time.

### 3 Scheduling Procedure

In order to solve (LP2), we propose the procedure reported in the pseudo-code from Algorithm (1) and explained next. The algorithm proceeds in the sense of the sequence and checks that, for each encountered node \( i \in \{1, \ldots, M\} \), a minimal total waiting time \( \Delta_i \) has been assigned up to node \( i \). If the algorithm detects a total waiting time shortage at node \( i \), i.e. \( \sum_{k=1}^{i} w_k < \Delta_i \), the total waiting time at \( i \) and its preceding nodes needs to be increased. Given that the objective function depends on the total vehicle load \( L_i \), the algorithm starts by considering adding waiting time to nodes \( j \leq i \) featuring the lowest minimal total load up to \( i \), i.e. \( j = \arg\min_{k \in \{1, \ldots, i\}} L_k \). Note that, if multiple nodes featuring an equivalent minimal total load exist, the first node can be selected without loss of generality. For node \( j \), the total waiting time can be feasibly increased by a maximum amount \( \delta w_j \) defined by constraints (15) and (16). That is, while deciding upon an increment in waiting time at node \( j \), one needs to check that excess-ride time constraints are not violated for requests whose pickups precede \( j \) and whose dropoffs follow \( j \). Furthermore, in order to guarantee time-window feasibility of the whole sequence, one needs to check that an increment in waiting time at \( j \) does not exceed the maximal waiting time that can be assigned to up to node \( j \), i.e. \( \Theta_j - \sum_{k=1}^{j} w_k \). Finally, waiting time at node \( j \) can be incremented by \( \delta w_j \), which is computed as the minimum between the amount defined by excess-ride time constraints, time-window constraints, and the total waiting time shortage defined by \( \Delta_i - \sum_{k=1}^{i} w_k \). After the update of \( w_j \), if node \( j \) has reached its maximum waiting time limit by updating \( \Delta_i \), that is \( \delta w_j < \Delta_i - \sum_{k=1}^{i} w_k \), node \( j \) is removed from the list \( \Omega \) of potential nodes whose waiting time may be further increased. If \( \sum_{k=1}^{i} w_k < \Delta_i \), i.e. there is still a total waiting time shortage at \( i \), the total waiting time is increased at the next node \( j \) up to \( i \) featuring the second lowest total vehicle load. This iterative process terminates as soon as \( \sum_{k=1}^{i} w_k \geq \Delta_i \), that is when sufficient waiting time has been assigned to \( i \) and all of its preceding nodes. In this case, the algorithm moves inspecting \( i+1 \) and up to the end of the sequence. If at the end of the whole process, \( \Delta_M \) has been feasibly assigned, the algorithm terminates with a feasible solution. Section 3.1 shows that if a feasible solution is not obtained, this may lead to an incorrect infeasibility declaration due to maximum ride time violations. On these solutions, it is possible to recover feasibility through the
recourse heuristic presented in Section 3.2, which is applied at step 17. in Algorithm (1). Note that the procedure between line 1 and 15 of Algorithm (1) results in a worst-time complexity of $O(M^2)$, since in the worst case, the step reported in line 6 may be executed $M$ times and the procedure in line 8. contains at most $M$ components. However, the application of the recourse heuristic leads to a worst-time complexity of $O(n \times M^2)$, since, Algorithm (1) may be restarted as many times as the number of requests $n$. Finally, note that Algorithm (1) can either terminate with a feasible solution, a correct infeasibility declaration, a suboptimal solution, or with an incorrect infeasible solution, on which feasibility can be heuristically recovered.

**Algorithm 1: Ride time-oriented scheduling algorithm**

| Input: | vehicle route sequence ($I = \{1, \ldots, M\}$), pickups $P_i$, dropoffs $D_i$, earliest start times $ET_i$, latest start times $LT_i$, maximum ride times $u_P$, service durations $s_i$, travel times $t_{ij}$ |
| Output: | Waiting times $w_i$, with $i \in I$, feasibility check |
| 1 | initialize $w_i = 0 \ \forall i \in \{1, \ldots, M\}$; |
| 2 | initialize $\Omega = \emptyset$; |
| 3 | initialize start node = 1; |
| 4 | initialize check = true; |
| 5 | while check = true do |
| 6 | for $i \rightarrow$ start node; $M$ do |
| 7 | Update: $\Omega = \Omega \cup \{i\}$; |
| 8 | while $\sum_{k=1}^{i} w_k < \Delta_i$ do |
| 9 | Set: $j = \arg\min_{L_k \in \Omega} L_k$; |
| 10 | Compute: $\delta w_j = \min_{k \in \{P | k \leq j \text{ and } n+k \geq j\}} u_k - \sum_{l=1}^{(n+k)-1} t_{l,j+1} - \sum_{l=1}^{(n+k)-1} d_l - \sum_{l=1}^{n+k} w_l; \quad \Theta_j = \sum_{k=1}^{i} w_k; \quad \Delta_i = \sum_{k=1}^{i} w_k$; |
| 11 | Set: $w_j = w_j + \delta w_j$; |
| 12 | if $\delta w_j < \Delta_i - \sum_{k=1}^{i} w_k$ then |
| 13 | Update: $\Omega = \Omega \setminus \{j\}$; |
| 14 | if $\sum_{k=1}^{i} w_k \geq \Delta_i$ then |
| 15 | break; |
| 16 | if $\sum_{k=1}^{i} w_k < \Delta_i$, & $\Omega = \emptyset$ then |
| 17 | Employ: Recourse heuristic (Algorithm 2); |
| 18 | if Recourse heuristic is not successful then |
| 19 | check = false; |
| 20 | break; |
| 21 | else |
| 22 | start node = $j + 1 \leq i$; |
| 23 | Update waiting times; |
| 24 | Restart algorithm from line 6. |

**3.1 Incorrect infeasibility declarations**

There may be cases in which, at the end of Algorithm (1), the total waiting time shortage $\Delta_{i,j}$ cannot be feasibly assigned. In particular, it is possible that the waiting times at all nodes preceding $i$ have been updated, i.e., $\Omega = \emptyset$, but there is still a shortage of waiting time at node $i$, i.e., $\sum_{k=1}^{i} w_k < \Delta_i$. In this case, the Algorithm (1) prematurely terminates and the sequence is deemed infeasible. However, note that, due to the myopic nature of the procedure, this may result in an incorrect infeasibility declaration. In fact, the procedure is designed to assign waiting times to nodes $j \leq i$ based on the waiting times shortage $\Delta_i$ and without considering the waiting times shortages at subsequent nodes $\{i + 1, \ldots, M\}$. To ensure feasibility at subsequent nodes after detecting a waiting times shortage $\Delta_i$, it may be necessary to apply waiting times to nodes $j \leq i$ that: 1. are not the first node among nodes featuring the lowest total load; and 2. do not feature the lowest total load. Failing to apply waiting times at the right nodes without knowledge of future waiting times shortages may result in situations in which the nodes in $\Omega$ cannot be further pushed forward in time without violating the maximum ride time constraints of nodes in $\Omega$ and, consequently, an incorrect infeasibility declaration.

This drawback is illustrated in the routing sequence in Figure 1 and explained next. Suppose that, at iteration ten, Algorithm (1) detects a waiting time shortage $\Delta_{P_{10}} > 0$ and that all of this waiting time shortage is applied at pickup node $P_{10}$, given that it features the lowest total load among all of its preceding nodes. As a consequence of increasing the waiting times at node $P_{10}$, suppose that the service start time at node $D_6$ reaches the latest service start time $dep_{D_6}$.
and that the ride time of node $P_7$, with dropoff location $D_7$, reaches its maximum. The algorithm proceeds inspecting the sequence and a new waiting times shortage is detected at node $P_{10}$, i.e., $\Delta_{P_{10}} > 0$. However, it is not possible to increase waiting times at nodes preceding $P_{10}$, given the time window constraints at node $D_6$, and it is not possible to increase the waiting times at node $\Delta_{P_{12}}$, given the maximum ride time constraints at node $D_7$. In this case, the scheduling algorithm prematurely terminates and returns an infeasibility declaration for the given sequence. However, this is an incorrect infeasibility declaration given that a feasible solution can be constructed by applying the waiting time shortage $\Delta_{P_{10}}$ at node $P_7$ instead of node $P_{10}$, although the total load at node $P_7$ is higher than the total load at node $P_{12}$. This reduces the ride time of node $P_7$, with dropoff location $D_7$ following node $P_{12}$. Consequently, the waiting times shortage $\Delta_{P_{12}}$ can be applied at node $P_{12}$, without violating the maximum ride time constraints of node $P_7$ and a feasible solution is returned. To prevent incorrect infeasibility declarations, we propose a recourse heuristic, whose aim is to revisit previously-made waiting time decisions and decrease the chances of obtaining incorrect infeasibility declarations. The details of the recourse heuristic are provided in Algorithm 2. and are explained next.

### 3.2 Recourse heuristic to reduce incorrect infeasibility declarations

Suppose that the scheduling algorithm has detected a waiting times shortage $\Delta_i$ at node $i$ and that, after inspecting all nodes $j \leq i$, $\Delta_i > 0$ and $\Omega = \emptyset$. In this case, it may be necessary to modify waiting time decisions that were taken at steps preceding $i$ through a heuristic procedure, e.g., Algorithm 1. In particular, there may exist nodes that reached their maximum ride time as a consequence of waiting time decisions at previous nodes featuring the lowest total loads. Nodes $j \leq i$ that reached a maximum ride time can be detected by employing the first term in step 10. of Algorithm 1. Suppose that $M$ nodes $j_1 < j_2 < \ldots < j_i$ have reached their maximum ride time at iteration $i$. This means that, in order to recover a feasible route, the sum of the waiting time at these nodes need to be shifted by a total of $\Delta_i$. Note that postponing the service start time at $j_l$, $l \in \{1, \ldots, M\}$, automatically postpones the service start times at any successive nodes, including nodes $j_{2}, \ldots, j_i$. Furthermore, it may modify waiting time decisions at nodes following $j_l$. As such, the recourse heuristic iteratively explores possibilities to postpone service start times at nodes $j_l$, $l \in \{1, \ldots, M\}$, by employing the conditions defined by constraints (15) and (16) and employed in step 10. of Algorithm 1. Note that the proposed recourse algorithm is a heuristic, given that the waiting time shortage could be applied at nodes $j_l$ as well as nodes preceding $j_l$. If the waiting times at node $j_l$ can be postponed by a maximum amount $\delta w_{jl} > 0$, the waiting time at $j_l$ is updated, the waiting times at all nodes following $j_l$ are re-initialized, and Algorithm 1 is re-started from $j_{l+1}$. Otherwise, if $\delta w_{jl} = 0 \forall j_l$, $l \in \{1, \ldots, M\}$, then the solution is deemed infeasible and Algorithm 1 is prematurely terminated.

---

**Algorithm 2: Recourse algorithm**

**Input:** 
- vehicle route sequence ($I = \{1, \ldots, M\}$), pickups $P_i$, dropoffs $D_i$, earliest start times $ET_i$, latest start times $LT_i$, maximum ride times $u_{Pi}$, service durations $s_i$, travel times $t_{ij}$, waiting times $w_i$, remaining waiting time shortening $\Delta_i$. 

**Output:** 
- start node $start node$, waiting times $w_i$ with $i \in I$, feasibility check.

1. initialize $start node = 1$; 
2. initialize $check = false$; 
3. find all nodes $j_l \leq i, l \in \{1, \ldots, M\}$ that have reached maximum ride time constraints; 
4. for $l \rightarrow i : M$ do 
   5. Set: $\hat{w}_i = w_i, \hat{w}_l = 0 \forall l \in \{j_{l+1}, \ldots, i\};$ 
   6. Recompute: $LT_i$; 
   7. Compute: $\delta w_{jl}$ as in step 10. of Algorithm 1; 
   8. if $\delta w_{jl} > 0$ then 
      9. Update: $\hat{w}_{jl} = \hat{w}_{jl} + \delta w_{jl}$; 
      10. Update: $start node = j_l$; 
      11. Update: $check = true$; 
      12. Set: $\hat{w}_i = \hat{w}_i$; 
      13. break; 

Figure 1: Example of a route leading to an incorrect infeasibility declaration.
Algorithm 3: Battery management heuristic

Input: vehicle route sequence \( I = \{1, \ldots, M\} \), Waiting times \( w_i \) \( i \in I \), charging facilities \( s_1, \ldots, s_N \), charging rates \( \alpha_s \), discharging rate \( \beta \), travel times \( t_{i,i+1} \), earliest start times \( ET_i \), latest start times \( LT_i \), State of charge \( B_i \)

Output: Charging times \( E_s \) with \( s \in \{s_1, \ldots, s_N\} \), Boolean battery feasibility check

1. initialize \( \text{check} = \text{true} \);
2. initialize \( B_1 = B_0 \);
3. while \( \text{check} = \text{true} \) do
   for \( i \in {2, \ldots, M} \) do
   Compute: \( B_i = B_{i-1} - \beta \times t_{i,i} \);
   if \( B_i < 0 \) then
     the sub-sequence is infeasible \( \rightarrow \text{check} = \text{false} \);
   if \( i \in S \) then
   Set: \( \bar{E}_i = \min \{LT_{i+1} - ET_i; (Q - B_i)/\alpha_s\} \);
   Set: \( B_i = B_i + \bar{E}_i \);
   Update: \( w_i \) \( \forall i \in \{1, \ldots, M\} \);

4 Battery Management Problem

In the case of electric and autonomous vehicles, i.e., an e-ADARP, it is necessary to show that the retrieved excess-time optimal schedules are also battery-feasible. Indeed, Algorithm 1 disregards battery considerations which are instead part of e-ADARPs. Battery-feasibility aspects can be implemented by integrating battery-related decision variables and constraints into (LP1), as presented in Section 2.2 in [Bongiovanni et al., 2019], and summarized next.

Consider the predetermined route sequence \( I \) of \( M \) nodes. As described in Section 2, this route originates at an origin depot, terminates at a destination depot, visits a total of \( n \) requests, and may pass through \( N \) charging stations \( s \in S \subset I \). The latter are necessary to refill the battery level \( B_i \) of the vehicle after spending battery inventory at rate \( \beta_i \) for \( i \in \{1, \ldots, M - 1\} \). The vehicle can only access charging stations \( s \in S \) when no requests are onboard and is allowed recharge for a variable duration \( E_s \) at rate \( \alpha_s \). Furthermore, it departs from the origin depot with a pre-defined vehicle battery inventory \( B_{init} \) and is required to reach the destination depot with a minimal vehicle battery inventory \( B_{end} \). The latter is defined as a proportion of the total vehicle battery capacity \( Q \), by means of a minimal battery level ratio \( r \). Under these settings, the integrated scheduling and battery management problem is represented as (LP1), supplemented with the following constraints:

\[
B_i = B_{init} \quad i = 1
\]

\[
B_{i+1} \leq B_i - \beta_i \quad \forall i \in \{1, \ldots, M - 1\} \setminus S, i \neq j
\]

\[
B_{i+1} \geq B_i - \beta_i \quad \forall i \in \{1, \ldots, M - 1\} \setminus S
\]

\[
B_{s+1} = B_s + \alpha_s E_s - \beta_{s+1} \quad s \in S
\]

\[
Q \geq B_s + \alpha_s E_s \quad \forall s \in S
\]

\[
B_i \geq rQ \quad i = M
\]

\[
E_s \leq T_{s+1} - t_{s,s+1} - T_s \quad \forall s \in S
\]

\[
E_s \geq T_{s+1} - t_{s,s+1} - T_s \quad \forall s \in S
\]

\[
B_i \geq 0 \quad \forall i \in I
\]

\[
E_s \geq 0 \quad \forall s \in S
\]

4.1 Battery management heuristic

The battery management problem, defined by (LP1) supplemented with constraints (17)-(27), is solved through the battery management heuristic in Algorithm 3, which builds on the assumption that battery levels can be seen as an inventory which can only decrease with traveling. Then, for feasibility purposes, it is always better to recharge as much as possible, as early as possible. If the schedule that maximizes battery recharging does not satisfy the imposed battery management constraints, no other recharging plan may be feasible, i.e., the given sequence and excess-time optimal schedule is declared infeasible. This is due to the fact that the scheduling algorithm presented in Algorithm 1 is designed to minimize the completion time and, consequently, to provide the shortest schedules.
and the longest charging opportunities. Consequently, if battery feasible solutions do exist, at least one of them minimizes the total user excess ride time. Note that the proposed recharging procedure in Algorithm 3 is a heuristic since it cannot be formally proven optimal (e.g. maximizing the charging time at station $i$ may decrease opportunities to recharge more at following charging stations).

The battery management heuristic in Algorithm 3 considers the route schedule computed by employing Algorithm 1 and sequentially updates the battery inventory between successive nodes as a result of the initial battery level $B_i$, the battery discharge rate $\beta$, and the total travel time up to $i$. Note that battery discharge can be equally computed by energy consumption models (Goek and Schneider 2015, Pelletier et al. 2017). During this iterative process, when a charging facility is encountered, its maximal recharging time is bounded by: (i) the difference between the service start time at the current node and the latest service start time at the following node, and (ii) the time needed to fully recharge. To ensure feasibility, the recharging time at the station needs to be set to the minimum of the two. After the charging time is calculated, the station’s battery level and the waiting times of the station and subsequent nodes are updated. The procedure prematurely terminates only if a node $i$ features a negative battery inventory, after which the route is declared battery-infeasible.

### 5 Numerical Results

Numerical experiments are performed on the DARP instances presented in Cordeau and Laporte (2003) and on the e-ADARP instances presented in Bongiovanni et al. (2019). Specifically, we extract routing solutions by employing the adaptive large neighborhood search described in Ropke and Pisinger (2006), with 1,000 iterations. For each problem instance, the initial solution is constructed from the best known solution, if available, or by employing a construction heuristic in which a random subset of customers are assigned to the vehicles, and served in a taxi mode (i.e., pickup-dropoffs). At every iteration of the large neighborhood search, we solve the scheduling problem through a linear program and compare its results to the route evaluation procedures by Cordeau and Laporte (2003), Parragh et al. (2009), and Molenbruch et al. (2017b), as well as this paper. This results in a thorough comparison of the scheduling algorithms on 21,844,304 feasible DARP solutions and 2,370,243 feasible e-ADARP solutions. The numerical experiments are implemented in Julia v1.8.2 and run on 2 x AMD Rome 7352 @ 2.40 GHz 256M cache L3 CPU clusters. Each instance is run on five of such CPUs with 186 GB of RAM. The LP is implemented in the JuMP modeling language (Dunning et al. 2017) v1.3.1 and solved with Gurobi v 0.11.3.

Table 1 shows a comparison between the scheduling procedures on DARP instances. The first column indicates the instance name, following the convention employed in Cordeau and Laporte (2003). These are instances that consider three to 13 vehicles and 24 to 144 requests. For further information on the generation of these DARP instances, the reader is referred to Cordeau and Laporte (2003). The second column indicates the total number of evaluated feasible routes, and the third column the range of their size, i.e., the number of nodes in the routes. The fourth to six columns show the total number of incorrect infeasible declarations, the total number of deviating solutions, and their absolute average percentage deviation from the optimal solution, by employing Algorithm 1. The following columns show the same results for the 8-step scheduling algorithms by Cordeau and Laporte (2003), Parragh et al. (2009), and Molenbruch et al. (2017b). As it can be noted, Algorithm 1 does not return any incorrect infeasibility declaration. However, in some very rare cases (i.e., in only 27 occurrences over more than 21.5 million cases), it does return suboptimal solutions, namely for instances pr06, pr09, and, mostly, pr16. For these instances, the returned suboptimal solutions deviate by less than 2.5%, on average terms. This consists of a very small proportion of suboptimal solutions, compared to the results obtained by employing the scheduling algorithms by Cordeau and Laporte (2003), Parragh et al. (2009), and Molenbruch et al. (2017b). Consistently with reported results from the literature, the approach by Cordeau and Laporte (2003) has the tendency to minimize the total number of incorrect infeasibility declarations at the cost of a higher number of suboptimal solutions and their average deviations. Namely, the approach returns about 27% suboptimal solutions, with deviations between about 8% and 27%. The approach by Parragh et al. (2009), instead, has the tendency to minimize the total number of suboptimal solutions and their average deviation, at the cost of a higher number of incorrect infeasibility declarations. Namely, the approach returns up to 0.007% incorrect infeasibility declarations but reduces the number of suboptimal solutions to about 0.5%, with deviations between about 2.5% and 10%.
Table 2: Scheduling algorithms on DARP instances: CPU times [ms]

| Name   | pr01 | pr02 | pr03 | pr04 | pr05 | pr06 | pr07 | pr08 | pr09 | pr10 | pr11 | pr12 | pr13 | pr14 | pr15 | pr16 | pr17 | pr18 | pr19 | pr20 |
|--------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|        | 1.960 | 2.058 | 2.059 | 2.133 | 2.154 | 2.310 | 2.075 | 2.348 | 2.218 | 2.370 | 2.012 | 2.140 | 2.224 | 2.545 | 2.219 | 2.541 | 1.902 | 2.303 | 1.929 | 2.470 |
|        | 0.649 | 0.664 | 0.852 | 0.927 | 0.927 | 0.903 | 0.762 | 0.834 | 0.804 | 0.871 | 0.407 | 0.751 | 0.685 | 0.715 | 0.824 | 0.858 | 0.541 | 0.810 | 0.729 | 0.681 |
|        | 0.464 | 0.627 | 0.719 | 0.831 | 0.840 | 0.856 | 0.566 | 0.803 | 0.936 | 1.088 | 0.495 | 0.772 | 0.798 | 0.867 | 0.960 | 0.977 | 0.660 | 0.987 | 1.185 | 1.165 |
|        | 0.419 | 0.621 | 0.716 | 0.791 | 0.803 | 0.850 | 0.559 | 0.498 | 0.936 | 1.061 | 0.387 | 0.785 | 0.769 | 0.830 | 0.917 | 0.981 | 0.568 | 0.966 | 1.152 | 1.125 |
|        | 0.419 | 0.621 | 0.444 | 0.464 | 0.479 | 0.479 | 0.401 | 0.498 | 0.545 | 0.601 | 0.387 | 0.489 | 0.314 | 0.356 | 0.322 | 0.570 | 0.441 | 0.334 | 0.627 | 0.621 |

Table 3: Scheduling algorithms on E-ADARP instances: number of incorrect infeasible declarations, number of deviating solutions, and average relative deviation.

| Algorithm | 2003 | 2009 | 2017b |
|-----------|------|------|-------|
|           |      |      |       |
|           |      |      |       |
| u2-16-0-4 | 0    | 0    | 0     |
| u2-16-0-7 | 0    | 0    | 0     |
| u2-20-0-4 | 0    | 0    | 0     |
| u2-20-0-7 | 0    | 0    | 0     |
| u3-18-0-1 | 0    | 0    | 0     |
| u3-18-0-4 | 0    | 0    | 0     |
| u13-18-0-1| 0    | 0    | 0     |
| u19-18-0-1| 0    | 0    | 0     |
| u26-18-0-1| 0    | 0    | 0     |
| u26-18-0-7| 0    | 0    | 0     |
| u26-18-0-9| 0    | 0    | 0     |
| u30-18-0-1| 0    | 0    | 0     |
| u30-18-0-5| 0    | 0    | 0     |
| u5-50-0-7 | 0    | 0    | 0     |
| u5-50-0-9 | 0    | 0    | 0     |

Molenbruch et al. (2017b) produces similar results with respect to the procedure by Parragh et al. (2009), most specifically in terms of the number of incorrect infeasible declarations. However, the number of suboptimal solutions to 0.23%, with deviations between about 2% and 9.5%. It is worth noting that, in our numerical experiments, all incorrect infeasibility declarations returned by the scheduling algorithms by Parragh et al. (2009) and Molenbruch et al. (2017b) are due to violations of maximum ride time constraints. In addition to producing solutions of higher quality with respect to state-of-the-art scheduling algorithms, Algorithm (1) is computationally efficient, as shown in Table 2. It is about 60% faster than a linear program and comparably efficient with respect to Cordeau and Laporte (2003), Parragh et al. (2009), and Molenbruch et al. (2017b), with a slight computational disadvantage for small-scale instances, e.g., pr01, and advantage for large-scale instances, e.g., instances pr18, pr19, and pr20.

Table 3 shows a comparison between the investigated scheduling procedures on e-ADARP instances. The first column indicates the instance name, following the convention employed in Bongiovanni et al. (2019), i.e., <u><number of vehicles>-<number of customers>-<minimum battery inventory ratio at the destination depot>,”u” is used to refer to the instances adapted from real data from Uber Technologies Inc. Furthermore, the reader is reminded that the minimum battery inventory ratio is used to compute the minimal battery levels that all vehicles need to have at the destination depot, i.e. a minimal battery ratio of 0.7 means vehicles need to have at least 70% of their nominal battery capacity at the destination depot. The following columns adopt the same convention used in Table 1. As it can be noted, in this case Algorithm (1) always returns optimal scheduling solutions, which include charging.
Table 4: Scheduling algorithms on E-ADARP instances: infeasibility reasons

| Name       | Cordeau and Laporte (2003) | Parragh et al. (2009) | Molenbruch et al. (2017) |
|------------|----------------------------|-----------------------|--------------------------|
|            | Inf. | TW | Batt | Inf. | TW | Batt | Inf. | TW | Batt |
| u2-16-0.1  | 0    | 0  | 0    | 0    | 181 | 0    | 0    | 0    | 0    |
| u2-16-0.4  | 0    | 0  | 0    | 63   | 0   | 3727 | 0    | 0    | 0    |
| u2-16-0.7  | 0    | 0  | 0    | 63   | 0   | 3727 | 0    | 0    | 0    |
| u2-20-0.1  | 0    | 0  | 0    | 11   | 0   | 1259 | 0    | 0    | 0    |
| u2-20-0.4  | 0    | 0  | 0    | 1059 | 0   | 2986 | 0    | 0    | 0    |
| u2-20-0.7  | 0    | 0  | 0    | 63   | 0   | 3727 | 0    | 0    | 0    |
| u3-16-0.1  | 0    | 0  | 0    | 1115 | 0   | 1532 | 0    | 0    | 0    |
| u3-16-0.4  | 0    | 0  | 0    | 3727 | 0   | 364  | 0    | 0    | 0    |
| u3-16-0.7  | 0    | 0  | 0    | 63   | 0   | 3727 | 0    | 0    | 0    |
| u3-20-0.1  | 0    | 0  | 0    | 10   | 0   | 757  | 0    | 0    | 0    |
| u3-20-0.4  | 0    | 0  | 0    | 10   | 0   | 757  | 0    | 0    | 0    |
| u3-20-0.7  | 0    | 0  | 0    | 10   | 0   | 757  | 0    | 0    | 0    |
| u5-50-0.1  | 0    | 0  | 0    | 10   | 0   | 677  | 0    | 0    | 0    |
| u5-50-0.4  | 0    | 0  | 0    | 3    | 0   | 1044 | 0    | 0    | 0    |
| u5-50-0.7  | 0    | 0  | 0    | 3    | 0   | 1044 | 0    | 0    | 0    |

Decisions. Differently from the DARP results shown in Table 1, the approach of Cordeau and Laporte (2003) return some incorrect infeasibility declarations, although this remains limited to about 0.05%. The number of suboptimal solutions, however, rise up to about 34% and so their absolute average deviations from the optimal solution, which range between about 24% and more than 100%. The approach of Parragh et al. (2009) also shows an increase in the total number of incorrect infeasibility declarations with respect to the DARP results shown in Table 1. Namely, the total number of infeasibility declarations rise to at most about 10%, however, the number of suboptimal solutions decrease to about 0.006%, with deviations which are contained between about 0.5% and 7%. Finally, the approach of Molenbruch et al. (2017) shows a net decrease both in terms of the total number of incorrect infeasibility declarations and suboptimal solutions, with a total of eight occurrences over more than 2 million cases, with respect to the DARP results shown in Table 1.

As shown in Table 4, in the case of an e-ADARP, the procedure by Cordeau and Laporte (2003) mostly produce solutions that violate ride time constraints, whereas the procedure by Parragh et al. (2009) mostly violate time window and battery constraints. This last procedure is in fact designed to minimize completion time, which, in turn, may increase the chances of violating battery management constraints. As for the DARP results shown in Table 1, the procedure by Molenbruch et al. (2017) may violate ride time constraints. Finally, Table 5 shows the average CPU times, in milliseconds, obtained by running the linear program, Algorithm 1, and the scheduling procedures by Cordeau and Laporte (2003), Parragh et al. (2009), and Molenbruch et al. (2017) on the e-ADARP. As it can be noted, also in this case, Algorithm 1 is about 60% faster with respect to a linear program, on average, and up to about 80-85% faster, e.g. for instance u5-40-0.7 and instances u4-16. Savings in terms of computing time might be even more significant when considering larger problem instances or when several static problems are solved in real time, e.g. as in the dynamic e-ADARP presented in Bongiovanni et al. (2022).

6 Summary

This work proposed an excess-ride-time procedure for scheduling dial-a-ride instances. The procedure is shown to produce high-quality solutions, which are only very rarely suboptimal, in at least half the time of a linear program and in comparable time with respect to state-of-the-art scheduling algorithms. The proposed procedure can be potentially employed to efficiently solve scheduling problems from static and dynamic met heuristic approaches. For the e-ADARP, we further propose a battery heuristic which can be used when the vehicle routes include one or more visits to charging facilities. The heuristic is needed to provide battery-feasible charging plans for the vehicle schedules. The battery heuristic builds on the assumption that charging as much as possible as early as possible is the best strategy for battery-feasibility.

Computational experiments are carried on a plethora of static DARP and e-ADARP instances from the literature, which are extracted from an adaptive large neighborhood search with 1,000 iterations. Experiments show that the route evaluation procedure is computationally more efficient than solving a linear program, while returning higher-quality solutions with respect to popular scheduling heuristics from the literature. Furthermore, results show that the proposed scheduling procedure does not produce incorrect infeasibility declarations and produces suboptimal solutions in very rare cases (i.e., in about 0.0001 % of times, which deviate up to 2.5% from optimal solutions, on average).
Table 5: Scheduling algorithms on E-ADARP instances: CPU times [ms]

| Name     | Linear Program | Cordeau and Laporte (2003) | Parragh et al. (2009) | Molenbruch et al. (2017b) |
|----------|----------------|----------------------------|-----------------------|---------------------------|
| u2-16-0-4| 1.787          | 0.663                      | 0.471                 | 0.452                     | 0.499                     |
| u2-16-0-7| 1.966          | 0.503                      | 0.392                 | 0.359                     | 0.596                     |
| u2-20-0-4| 2.082          | 0.936                      | 0.661                 | 0.634                     | 0.484                     |
| u2-20-0-7| 2.655          | 1.023                      | 0.726                 | 0.738                     | 0.539                     |
| u2-20-0-9| 2.099          | 0.774                      | 0.560                 | 0.554                     | 0.520                     |
| u2-24-0-1| 2.872          | 1.034                      | 0.658                 | 0.649                     | 0.629                     |
| u2-24-0-4| 3.013          | 0.939                      | 0.657                 | 0.649                     | 0.697                     |
| u2-24-0-7| 2.558          | 1.110                      | 0.976                 | 0.941                     | 1.771                     |
| u3-16-0-1| 2.008          | 0.615                      | 0.577                 | 0.565                     | 0.355                     |
| u3-18-0-1| 2.079          | 0.607                      | 0.545                 | 0.534                     | 0.353                     |
| u3-18-0-2| 1.786          | 0.578                      | 0.533                 | 0.547                     | 0.383                     |
| u3-24-0-1| 2.088          | 0.722                      | 0.459                 | 0.468                     | 0.389                     |
| u3-24-0-4| 2.305          | 0.741                      | 0.495                 | 0.496                     | 0.434                     |
| u3-24-0-7| 2.756          | 0.758                      | 0.529                 | 0.515                     | 0.488                     |
| u3-30-0-1| 2.365          | 1.235                      | 0.758                 | 0.749                     | 0.453                     |
| u3-30-0-4| 2.954          | 1.238                      | 0.761                 | 0.750                     | 0.540                     |
| u3-30-0-7| 2.486          | 1.042                      | 0.623                 | 0.572                     | 0.551                     |
| u3-36-0-1| 2.562          | 1.418                      | 0.912                 | 0.885                     | 0.526                     |
| u3-36-0-4| 2.091          | 1.191                      | 0.755                 | 0.704                     | 0.505                     |
| u3-36-0-7| 2.365          | 1.329                      | 0.829                 | 0.836                     | 0.616                     |
| u4-16-0-1| 1.394          | 0.279                      | 0.194                 | 0.193                     | 0.247                     |
| u4-16-0-4| 1.404          | 0.283                      | 0.187                 | 0.185                     | 0.253                     |
| u4-16-0-7| 1.478          | 0.286                      | 0.249                 | 0.247                     | 0.337                     |
| u4-24-0-1| 1.824          | 0.627                      | 0.329                 | 0.328                     | 0.333                     |
| u4-24-0-4| 1.792          | 0.663                      | 0.379                 | 0.366                     | 0.373                     |
| u4-24-0-7| 1.731          | 0.674                      | 0.417                 | 0.414                     | 0.415                     |
| u4-32-0-1| 1.958          | 0.806                      | 0.522                 | 0.513                     | 0.415                     |
| u4-32-0-4| 1.875          | 0.848                      | 0.500                 | 0.485                     | 0.428                     |
| u4-32-0-7| 2.076          | 0.836                      | 0.472                 | 0.497                     | 0.421                     |
| u4-40-0-1| 2.216          | 1.064                      | 0.643                 | 0.624                     | 0.486                     |
| u4-40-0-4| 2.031          | 0.918                      | 0.541                 | 0.535                     | 0.488                     |
| u4-40-0-7| 1.499          | 0.856                      | 0.273                 | 0.261                     | 0.383                     |
| u4-48-0-1| 2.445          | 1.339                      | 0.990                 | 0.964                     | 0.543                     |
| u4-48-0-4| 1.647          | 0.487                      | 0.345                 | 0.339                     | 0.400                     |
| u4-48-0-7| 1.605          | 0.414                      | 0.340                 | 0.345                     | 0.405                     |
| u5-40-0-1| 1.945          | 0.849                      | 0.515                 | 0.494                     | 0.390                     |
| u5-40-0-4| 2.145          | 0.719                      | 0.543                 | 0.532                     | 0.436                     |
| u5-40-0-7| 1.325          | 0.284                      | 0.199                 | 0.195                     | 0.299                     |
| u5-50-0-1| 2.185          | 1.057                      | 0.645                 | 0.629                     | 0.429                     |
| u5-50-0-4| 2.206          | 1.084                      | 0.692                 | 0.678                     | 0.461                     |
| u5-50-0-7| 2.306          | 1.085                      | 0.702                 | 0.721                     | 0.507                     |

Acknowledgments

This research was enabled in part by support provided by Calcul Québec (www.calculquebec.ca/en/) and Compute Canada (www.computecanada.ca).

References

Bongiovanni, C., Kaspi, M., Cordeau, J.F., Geroliminis, N., 2022. A machine learning-driven two-phase metaheuristic for autonomous ridesharing operations. Transportation Research Part E: Logistics and Transportation Review 165, 102835.

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

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

Cordeau, J.F., Laporte, G., 2007. The dial-a-ride problem: models and algorithms. Annals of Operations Research 153, 29–46.

Dunning, I., Huchette, J., Lubin, M., 2017. Jump: A modeling language for mathematical optimization. SIAM Review 59, 295–320.

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

Gschwind, T., Drexl, M., 2019. Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. Transportation Science 53, 480–491.

Molenbruch, Y., Braekers, K., Caris, A., 2017a. Typology and literature review for dial-a-ride problems. Annals of Operations Research 250 (1-2), 295–325.

Molenbruch, Y., Braekers, K., Caris, A., Bergh, G.V., 2017b. Multi-directional local search for a bi-objective dial-a-ride problem in patient transportation. Computers & Operations Research 77, 58–71.

Parragh, S.N., Doerner, K.F., Hartl, R.F., Gandibleux, X., 2009. A heuristic two-phase solution approach for the multi-objective dial-a-ride problem. Networks: An International Journal 54, 227–242.

Pelletier, S., Jabali, O., Laporte, G., Veneroni, M., 2017. Battery degradation and behavior for electric vehicles: Review and numerical analyses of several models. Transportation Research Part B: Methodological 103, 158–187.
Savelsbergh, M.W., 1992. The vehicle routing problem with time windows: Minimizing route duration. ORSA Journal on Computing 4, 146–154.

Vidal, T., Crainic, T.G., Gendreau, M., Prins, C., 2015. Timing problems and algorithms: Time decisions for sequences of activities. Networks 65, 102–128.

Yue, S., Dupin, Nicolas, Puchinger, J., 2023. A deterministic annealing local search for the electric autonomous dial-a-ride problem. European Journal of Operational Research.

Yue, S., Dupin, Nicolas, Parragh, S., Puchinger, J., 2022. A Column Generation Approach for the Electric Autonomous Dial-a-Ride Problem. arXiv preprint arXiv:2206.13496.