Charging management of shared taxis: Neighbourhood search for the E-ADARP

Toussaint Hoché, Dominique Barth, Thierry Mautor, Wilco Burghout

Charging management of shared taxis: Neighbourhood search for the E-ADARP

Toussaint Hoché[†,‡], Dominique Barth[†], Thierry Mautor[†] and Wilco Burghout[‡,*]

Abstract— The electric vehicle market is booming. However, these vehicles need to be refilled more often and do so much more slowly than internal combustion engine (ICE) vehicles. The arrival of autonomous vehicles will enable both fully centralised systems for taxi fleet management and a 24/7 use of each taxi. Finally, the ride-sharing market is also booming. Thus, efficient future taxi fleets will have to provide efficient, integrated solutions for ride-sharing, charging and automation. In this paper, the problem focused on is a variation of the Dial-A-Ride-Problem (DARP) where charging as well as the availability of charging stations are taken into account: Given a fleet of autonomous and electric taxis, a charging infrastructure, and a set of trip requests, the objective is to assign trips and charges to taxis such that the total profit of the fleet is maximised. Our contribution consists in the development of a greedy method, and of a simulated annealing. Our methods are evaluated on large instances (10000 requests) based on taxi trip datasets in Porto. Our conclusions show that while high-capacity batteries are largely unneeded in normal circumstances, they are capital in case of disruption, and useful when the charging infrastructure is shared, with queueing time to access to a charger. Parking searches also represent a significant energy expense for autonomous taxis.

I. Introduction

Internal combustion engine (ICE) vehicles are being phased-out in multiple countries. Electric vehicles are considered as an alternative, however, their range is more limited and refilling them is currently much slower: A gasoline pump provides around 500km of range per minute, while a fast charger gives around 10km per minute[1]. A charger as fast as a gas pump would be a 5MW charger. A nuclear reactor could only handle around 200 of these "electric gas pumps". Therefore, the number of charging stations and their speed may be limited. Moreover, their usage may be restricted: Electric taxis may be prohibited from using at peak hours of electric consumption (e.g. often around 18:00). Such limitations may affect negatively the income of electric taxi fleet owners, since their vehicles have little downtime. Also, autonomous taxis will likely have lower operating costs than conventional ones[1], particularly at night. This opens up new markets for taxis, which could be used, for instance, for food delivery or night-time deliveries, which would mean, in turn, a massive reduction of their downtime. Finally, the ride-sharing market is booming. Fleets of shared vehicles can be much more efficient at time when the number of customers is very high, allowing the same number of customers to be treated by a smaller fleet. This can lead to a further reduction of the average downtime.

Optimizing the actions performed by a fleet of shared vehicles is generally modelised as a DARP (Dial A Ride Problem). The DARP is a generalisation of the VRP (Vehicle Routing Problem): a fleet of vehicles is dispatched to transport customers. Each customer has a request, characterised by an origin, a destination, a pick-up time, a tolerance for lateness and a number of seats. Customers accept that some detours occur during their trip, as long as the total duration of the detours remains below a threshold. The goal is to generate routes for the vehicles to maximise some objective function, generally, the number of satisfied customers. The literature on the DARP is prolific [2], [3]. However, battery management is a new research question [4]. Among the papers considering a variation of the VRP with recharging, some do not consider ride-sharing [5], [6], [7], [8], [9], [10], partial charges [8], [9], time windows [5], [10]. To the best of our knowledge, no paper mention any constraints on the availability of the chargers. Also, none of these papers seems to consider the impact of parking place searches, even though their impact on consumption may be significant. Our problem is a variant of the E-ADARP (Electric-Autonomous DARP) of [11]. In the E-ADARP, the vehicles have a limited battery and may use charging stations. In our problem, we introduce a new constraint: at some hours, chargers are unavailable, and a given charger can only be used by one taxi at a time. We also modelise on-street parking place searches. One motivation is that autonomous taxis may have a harder time finding a place to stop without blocking traffic completely. Data on on-street parking is scarce: For instance, [12] estimates that there are between 105 millions and 2 billions on-street parking places in the United States.

Since our problem generalises the VRP, it follows that it is NP-complete. In the literature, most evaluations of heuristics have been performed on small instances (generally, variants of Solomon's benchmark [13]). Fairly little attention is given to highly scalable methods: [11]

[†]Toussaint Hoché, Dominique Barth & Thierry Mautor are with the University of Versailles Saint-Quentin-en-Yvelines, Versailles, France.

[‡]Toussaint Hoché & Wilco Burghout are with the VEDECOM Institut, Versailles, France.

[*]Wilco Burghout is with the KTH Royal Institute of Technology, Stockholm, Sweden.

[1]Assuming a speed of 36L/min at the gasoline pump, a consumption of 7.2L/100km for the ICE car, a speed of 108kW for the charger, and a consumption of 18kWh/100km for the electric car.

consider at most 5 vehicles and 50 requests, and [6] at most 4 vehicles and 100 requests.

The DARP and its variants are generally solved exactly with a MILP (Mixed Integer Linear Program) and some optimizations, like a branch-and-price [9], [14], [10], which is not scalable. When heuristics are used, they generally are neighbourhood searches [8], [9], [10] inspired by the one used on similar problems. For very large instances, heuristics used for the dynamic version of the problem, in which requests arrive in real-time, may perform well while having a very good computation speed.

To solve our specific problem on very large instances (dozens of vehicles, thousands of requests), we propose a two steps heuristic. First, a greedy heuristic generates a solution. Starting from a solution in which taxis stay parked, requests are inserted one-by-one using a low-complexity algorithm, suitable for a dynamic algorithm. Then, the solution is improved with a local search. This local search performs moves such as charge insertion or the transfer of a request from a taxi to another.

II. Model

This section presents a MILP formulation of the problem.

T is defined as the set of taxis. Each taxi $t \in T$ has a passengers capacity $q_t \in \mathbb{N}^+$, a consumption per second spent driving $\delta_t^{time} \in \mathbb{R}^+$, a consumption per metre $\delta_t^{metre} \in \mathbb{R}^+$. The consumption per second is caused by the A/C, the sensors, and the inboard computer.

Let $G = \{V, A\}$ an oriented graph. Each vertex represents an action (pick/drop a customer, use a charger, leave the starting position, permanently return to the depot). Thus, a schedule of a taxi can be defined as a path in G respecting some constraints.

Let $V = \{B \cup \{\text{depot}\} \cup S \cup C\}$, such that:

- B is the set representing the beginning of the schedule of each taxi. $|B| = |T|$.
- depot is a vertex representing the end of the schedule of all taxis. It is a sink.
- S is the set of all stops. It is divided in two parts. S^+ represents the set of pick-ups, while S^- represents the set of drop-offs.
- C is the set of charges.

Each vertex $v \in V$ has a time window $[\alpha_v, \omega_v]$. The beginning of the schedule of the taxi $t \in T$ is noted $b_t \in B$. It is such that $\nexists t, t' \in T \mid t \neq t'$ such that $b_t = b_{t'}$.

Let R the set of all requests. Each request $r \in R$ has a seat demand $q_r \in \mathbb{N}^+$, a pick-up $p_r \in S^+$ and a drop-off $d_r \in S^-$. The unique request associated to the stop $s \in S$ is noted r_s. The time windows of p_r and d_r indicates the ideal pick-up and drop-off time of the customer and his tolerated lateness.

Let *Available* the chronologically ordered set of time windows during which chargers are available. $C^{j,k} \subset C$ denotes the set of all charges happening at the charger j during the k^{th} element of *Available*. Each charger j has a charge speed $speed_j \in \mathbb{R}^+$.

Each arc $a \in A$ has a length $l_a \in \mathbb{R}_{[0,+\infty[}$ and a duration $\tau_a \in \mathbb{N}_{[0,+\infty[}$. The arc going from $u \in V$ to $v \in V$ is noted $a = (u, v) \in A$.

The arc (u, v) is in A if and only if one of the following statements is true:

- $u \in B$ and $v \in S^+ \cup \{\text{depot}\}$
- $u \in S^+$ and $v \in S \backslash \{u\}$
- $u \in S^-$ and $v \in V \backslash \{B\}$ and, if $v \in S$, $r_u \neq r_v$
- $u \in C$ and $v \in \{\text{depot}\} \cup S^+$

depot is virtual, it just indicates the end of the schedules, i.e. Arcs going to depot have a null length and duration.

A. Output

The output is a set $X \in \{0,1\}^{|A|}$, which determines for each arc if it is used or not. The used arcs form $|T|$ paths going from each vertex of B to depot. Each path represents a taxi's schedule. The decision variable indicating if the arc $a \in A$ is used is noted x_a. The objective function is :

$$\max \sum_{r \in R} q_r t_{(p_r, d_r)} \sum_{u \in V} x_{(u, p_r)} \tag{1}$$

i.e. The total value of the requests treated is maximised, with the value of a request being its number of passengers $\times$ the duration of the arc from the origin to the destination of the request. Also, note that for the sake of brevity, we note

$$\sum_{u \in V} x_{(u,p_r)} \text{ instead of the correct } \sum_{u \in V | (u,p_r) \in A} x_{(u,p_r)}$$

B. Constraints

Some constraints are not written in a way considered valid for a MILP, however, some classical linearisation techniques that are not detailed in this paper can make them linear.

1) Flow: A vertex can only be visited once (except depot).

$$\sum_{v \in V} x_{(u,v)} \leq 1 \qquad \forall\ u \in V \backslash \{\text{depot}\} \tag{2}$$

If an arc (u, v) is used, then there must be a used arc (v, w).

$$\sum_{u \in V} x_{(u,v)} = \sum_{w \in V} x_{(v,w)} \qquad \forall\ v \in V \backslash \{B, \text{depot}\} \tag{3}$$

There is a used outgoing arc for each schedule's beginning (4). Every taxi schedule finishes with depot (5).

$$\sum_{v \in V} x_{(b_t,v)} = 1 \qquad \forall\ t \in T \tag{4}$$

$$\sum_{u \in V} x_{(u,\text{depot})} = |T| \tag{5}$$

The variable indicating if the taxi $t \in T$ goes through the vertex $v \in V \backslash \{\text{depot}\}$ is noted $y_v^t \in \{0,1\}$.

We note $Y = \{y_v^t \mid \forall\ v \in V \backslash \{\text{depot}\}, \forall\ t \in T\}$. Also, $\forall\ t \in T$, $\forall\ b \in B, y_b^t = 1$ if and only if $b = b_t$.

Taxis follow disjoints paths (except at the depot). i.e. if an arc (u,v) is used and the taxi t reach the vertex u, then it is necessarily the taxi t which reaches the vertex v by going through the arc (u,v).

$$y_v^t = \sum_{v \in V} y_u^t x_{(u,v)} \qquad \forall\ u \in V \backslash \{\text{depot}\}, \forall\ t \in T \tag{6}$$

2) Capacity: The variable indicating the number of clients inside the taxi going through the vertex $v \in V$ is noted $q_v \in \mathbb{N}^+$. We note $Q_v = \{q_v \mid \forall\ v \in V\}$. Taxis start empty and finish empty, and cannot charge while containing a client.

$$q_v = 0 \qquad \forall\ v \in B \cup C \cup \{\text{depot}\} \tag{7}$$

When a pick-up is reached, the number of passengers inside the taxi increases (8). Conversely, it decreases when a drop-off is reached (9). It does not change when a charge or the end of the schedule is reached (10). $\forall\ u \in V$:

$$q_{p_r} = q_u + q_r \qquad \forall\ r \in R \qquad \text{if } x_{(u,p_r)} = 1 \tag{8}$$

$$q_{d_r} = q_u - q_r \qquad \forall\ r \in R \qquad \text{if } x_{(u,d_r)} = 1 \tag{9}$$

$$q_u = q_v = 0 \qquad \forall\ v \in C \cup \text{depot} \qquad \text{if } x_{(u,v)} = 1 \tag{10}$$

A taxi cannot contains more customers than it has seats.

$$q_v \leq q_t \qquad \begin{array}{c} \forall\ v \in V, \\ \forall\ t \in T, \\ \text{if } y_v^t = 1 \end{array} \tag{11}$$

3) Time: $arrive_v$ is the time at which a vertex $v \in V$ is reached. $depart_v$ is the time at which a vertex $v \in V$ is left. The taxis start moving at time 0.

$$arrive_b = 0 \qquad \forall\ b \in B \tag{12}$$

A vertex must be reached before being left and its time window must be respected.

$$\alpha_v,\ arrive_v \leq depart_v \leq \omega_v \qquad \forall\ v \in V \tag{13}$$

A fixed amount of time is required to go through an arc.

$$depart_u + \tau_{(u,v)} = arrive_v \qquad \begin{array}{c} \forall\ u \in V, \\ \forall\ v \in V \backslash \{\text{depot}\} \\ \text{if } x_{(u,v)} = 1 \end{array} \tag{14}$$

A taxi must reach a customer's pick-up before his drop-off.

$$depart_{p_r} \leq depart_{d_r} \qquad \forall\ r \in R \tag{15}$$

The "parking phase" of $v \in V$ is what happens between its arrival and departure. A parking phase starts with a search phase, whose duration is noted $search_v \in \mathbb{N}$. The search phase ends after a duration $search_{max}$ at most.

$$search_v = \min(search_{max},\ depart_v - arrive_v) \tag{16}$$

4) Battery: $w_v \in \mathbb{R}_{[0,+\infty[}$ is the battery of a taxi when it reaches a vertex $v \in V$. A taxi starts with a battery level w_t^0. Reminder: $b_t \in B$ is the beginning of the schedule of the taxi $t \in T$ and w_t is its maximum battery.

$$w_{b_t} = w_t^0 \qquad \forall\ t \in T \tag{17}$$

A battery cannot be overfull nor empty.

$$0 < w_v \leq w_t \qquad \begin{array}{c} \forall\ v \in V \backslash \{\text{depot}\}, \\ \forall\ t \in T, \\ \text{if } y_v^t = 1 \end{array} \tag{18}$$

Reminder: l_a and τ_a are the length and duration of the arc $a \in A$, respectively, and δ_t^{time} and δ_t^{metre} are the consumptions per second and per metre of the taxi $t \in T$, respectively.

A taxi $t \in T$ crossing an arc $a \in A$ spends an amount of energy $\delta_a^t = -(l_a \delta_t^{metre} + t_a \delta_t^{time})$.

When searching for a parking place, taxis go at a constant speed $cruise\_speed \in \mathbb{R}^+$. The energy consumption at each vertex v is: $\delta_v^t = -search(v) \times (\delta_t^{time} + cruise\_speed \delta_t^{metre})$.

The energy loss between the departure from a vertex $u \in V$ and the departure from the vertex v is noted $\Delta_{(u,v)}$ such that:

$$\Delta_{(u,v)}^t = \delta_v^t + \delta_{(u,v)}^t \qquad \forall\ u,v \in V, \forall\ t \in T$$

This energy loss is the variation of energy for non-charge vertices:

$$w_u + \Delta_{(u,v)}^t = w_v \qquad \begin{array}{c} \forall\ u \in V \backslash \{C\}, \\ \forall\ v \in V \backslash \{\text{depot}\}, \\ \forall\ t \in T, \\ \text{if } x_{(u,v)} = 1 \end{array} \tag{19}$$

Reminder: c_i^j is the vertex representing the i-th use of the charger j and $speed_j$ is the speed of the charger j. The amount of energy recovered through a charge is Δ_i^j such that:

$$\Delta_i^j = speed_j \times (depart_{c_i^j} - search_{c_i^j} - arrive_{c_i^j}) \qquad \forall\ c_i^j \in C$$

The variation of energy after a charge is therefore:

$$w_{c_i^j} + \Delta_{(c_i^j,v)} + \Delta_i^j = w_v \qquad \begin{array}{c} \forall\ c_i^j \in C, \\ \forall\ v \in V \backslash \{\text{depot}\}, \\ \text{if } x_{(c_i^j,v)} = 1 \end{array} \tag{20}$$

A charger cannot be used by two taxis at once, i.e. A charge must finish before the next starts.

$$depart_{c_i^{j,k}} < arrive_{c_{i+1}^{j,k}} \qquad \forall\ i,j \in \mathbb{N} \tag{21}$$

III. Proposed Approach

Our approach consists in a greedy insertion of the requests, starting from an empty solution, followed by a simulated annealing.

A. Greedy

The greedy algorithm use 3 moves, called C^+,R^+ and R^+C^+. A ride is defined as a maximum sequence of stops (pick-up or drop-off) in a schedule, such that the taxi always contains at least one customer.

R^+ (Add request): A request q is inserted in a schedule s, while ignoring battery constraints, or nothing happens. Let $(r_1,\dots,r_n)$ the sequence of rides in s, ordered chronologically. First, we try to insert q in each ride. Inserting q in r_i means replacing r_i by a ride r' such that: 1) r' contains all the stops of r_i, in the same order. 2) r' contains the pick-up and drop-off of q. 3) There is no time constraint violation during the ride. 4) r' minimises the distance driven by the taxi during the ride. 5) The ride r_{i+1} does not need to be delayed, and the ride r_{i-1} does not need to be advanced. 6) In s, the vertices between r_{i-1} and r_{i+1} not in r_i can be removed. If it is possible, q is inserted in the ride and the move ends. If no such ride is found, then for any pair of rides (r_i,r_{i+1}), we try to insert the pick-up and drop-off of q between the end of r_i and the start of r_{i+1} (thus creating a new unshared ride). Neither r_i nor r_{i+1} can be advanced/delayed, but the vertices between them in s can be modified. q is inserted if it is possible, and the move ends. The move does nothing if no such pair of ride is found.

C^+ (Add charges): Charges are inserted in a schedule at the earliest possible time. First, we identify the intervals of time when the taxi is available (not in a ride). Second, during each interval, in chronological order, and for each charger, we check how much energy can be refilled. If no charger is able to give at least 1% more than the energy spent to reach the charger, then the taxi remains parked. Else, we send the taxi to the charger maximising the energy at the end of the interval. With the charges now updated in this interval, we look at the second interval of time, and perform the same process.

R^+C^+ (Add request, add charges): Uses the move R^+ followed by C^+.

The greedy works as follows: 1) Generate a solution in which every taxi immediately goes to the depot. 2) The set of requests R is ordered (by request's value, or by starting time, or randomly, ascendingly or descendingly). 3) For each request r in R, for each schedule s, we try to insert r in s with R^+C^+. If this move fails for every schedule, r is rejected. If this move succeeds for at least one schedule, then one of the valid schedule is chosen (the closest taxi is sent, or the one with the least/most battery, or a random one, or the one who has been idle the longest). When every request has been either inserted or rejected, the greedy insertion algorithm terminate.

B. Annealing

Four additional moves are defined for the annealing.

R^- (Remove request) Remove a request from a schedule. Its pick-up and drop-off are removed, and the departure and arrival time at the vertices of s are adjusted.

R^{swap} (Swap request): Swap requests between schedules. 1) Two requests q_1 and q_2 are removed from schedules s_1 and s_2, respectively, with the move R^-. 2) R^+C^+ is used on s_1 and q_2. 3) R^+C^+ is used on s_2 and q_1.

$R^{+-}C^+$ inserts a request q in a schedule s, and may remove rides. It first execute R^+C^+, and ends if it succeeds. Then, it tries to insert a new ride r treating only q. This ride starts at the ideal pick-up time of q. Ride of s incompatible with r because of time constraints are removed. Then r is progressively delayed, making some removed rides compatible again, and some other incompatible. In the end, r is inserted in the position which minimises the total value of the request removed. Finally, C^+ is used to makes the schedule valid.

R^+_{move} inserts a request q in a schedule s, part of a solution S using the greedy algorithm. First, q is inserted with R^{+-}, followed by C^+. If s is invalid, the move ends without changing anything. Else, the requests removed from s are reinserted in S, using the greedy algorithm.

In the remainder of this paper, "random" means "following an uniform distribution", unless stated otherwise. In the simulated annealing, each iteration, a random move is selected with a probability distribution chosen by the user. Inputs for the move chosen are chosen randomly: A schedule for C^+ ; A schedule and an untreated request for R^+,$R^{+-}C^+$ and R^+C^+ ; A schedule and a request it treats for R^-. For R^{swap}, two schedules and a request r treated by the first schedule are chosen randomly. Then, a request r' is chosen such that r' is in the second schedule, and r' is the first request treated after the start of r.

With the input parameters set, the move is executed. If it succeeds, let δ the variation of the value of the solution. If $\delta \geq 0$, the move is accepted. Else, it is accepted with a probability equal to $e^{\frac{\delta}{\text{temperature}}}$. The temperature decreases linearly as time passes, and the annealing finishes with a timer.

C. Complexity

In practice, R^+ and R^- have a complexity linear in the length of the schedule they are used on. In theory, in a case where n customers accept a very long lateness, a complexity in $O(n^2)$ can be reached. Inserting charges has a higher complexity: If a schedule contains n rides, and there are k chargers used u_c time each, then C^+ has a complexity of $O((n+u)k)$. In practice, this complexity can be reduced: Adding a charge before an instant t where the battery is full is useless if the battery never gets negative before t. Similarly, if the battery gets negative at t and no charge can be added before t, then the move fails, and there is no need to check anything after t.

IV. Experimentations

We implemented these algorithms in C++, and executed simulations on a laptop with a i7 processor. R^+C^+

is executed in parallel during the greedy.

A. Initialisation

To generate bigger instances, we first use the software SUMO (Simulation of Urban MObility) and the maps available on OSM (Open Street Map) to generate a graph G' representing Porto. An arc represents a portion of a roadway or a connection between roadways, and has an estimated speed and a length. An open dataset of 17000 taxi trips (kaggle.com) is used to generate the requests.

Each time a simulation is started, a set of 10000 requests is generated from trips chosen randomly. The origin/destination of the trips are presented as GPS locations, and we associate them to the closest arc of the graph. The requests are in July 2013, spread between Wednesday, 3rd at 00:00, and Sunday, 7th, at 24:00. The number of passengers is chosen randomly from 1 to 4. Customers accept at most 5 minutes of lateness at pick-up and drop-off, plus 20% of the expected ride duration.

There are 35 homogeneous taxis. They have 5 places, a battery of 72kWh. half are placed on random streets, and half are placed in the streets with the highest number of requests' pick-up. The taxis use 1.8kW for auxiliary system while driving, and 15kWh/100km at 10km/h (Each additional km/h increase the consumption by 1%). Their cruise speed when searching for a parking place is 10km, and they need 15 minutes to find a place. Taxis start the simulation with 60(±30)% of battery and must have at least 60% at the end. The chargers are placed with the method used for the taxis. Fast chargers have a speed of 72kW. Slow chargers have a speed of 7.2kW. 2 minutes are required to plug and unplug the charger, and wait that the previous user leaves.

Once chargers, taxis, and requests have been placed in G', we generate G. The distance and duration between two vertices of G are equal to the distance and duration of the path with the shortest duration in G'. Traffic is assumed constant (the street speed given by OSM is used).

B. Comparison between methods

The figure 1 presents the value of the solutions found by the different methods implemented. This value is indexed, such that 100 is the total value of the requests to treat (and an upper bound on the value of a solution), and for each method and each configuration, it is averaged over 3 simulations. Five methods are presented. The first letter indicates the criterion used to order the requests in the greedy: V = by value, S = start time. The second letter indicates the direction of this order: L = lowest first, H = highest first. The third letter indicates the criterion used to choose a taxi: C = closest, R = random, I = most idle (last ride's end before request's start is the lowest). If a method has + at the end, it means that it got improved by a simulated annealing. This annealing has an initial temperature of 500, and has at each iteration 50% chance to pick R^+_{move} (The

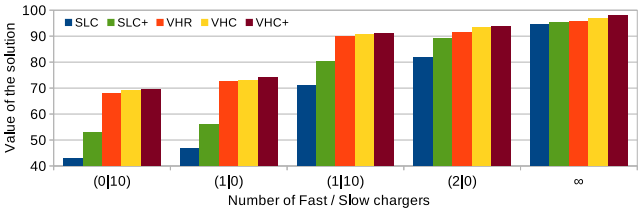

Fig. 1. Performances of the different algorithms in Porto on 5 different instances. The name of each instance, e.g. (0|24), indicates the number of fast and slow chargers, respectively. The instance ∞ has taxis with an unlimited battery.

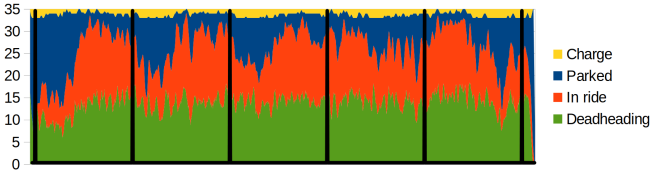

Fig. 2. Activities performed by the taxis in the instance (2|0). Every 7.5 minutes, a snapshot is taken and the number of taxis in each state is counted. The black vertical lines indicates midnight.

greedy used in this move is VHC), 10% chance to pick R^- and 40% chance to pick R^{swap}. The annealing lasts 15 minutes.

We tested every combination of the presented request orders (including random) and strategy (including sending the taxi with the least/most battery, or the least idle). The conclusions were that sending the closest taxi is consistently the best strategy, although it does not have a very significant impact on the outcome. Regarding the requests, ordering them by value from highest to lowest proved consistently better than any other strategies. The figure 2 show the activities performed by the taxi in the instance (2|0).

The simulated annealing offers some improvement to VHC: it multiply the value by around 1.03. Using different neighbour moves did not improve its efficiency. Replacing R^+_{move} by R^+C^+ does not have a significant impact: while neighbours are visited much faster, they have a lower chance to be accepted. We also performed experimentations in which charges should give at least 30% of battery to be considered. While preventing very small (and therefore inefficient) charges could have improved the solution, it actually worsened it.

C. Disruption and battery size

We evaluated the impact of battery size on the value of the solution when the charging infrastructure becomes unavailable (see figure 3). A very small battery size suffices in normal circumstances: 90% of requests can be treated with a 20kWh battery, and 80% with a 11kWh battery. However, the value of the solution begins to decrease linearly past a certain disruption duration. Making the disruption happens half a day sooner (finishes Friday at midday) produces similar results.

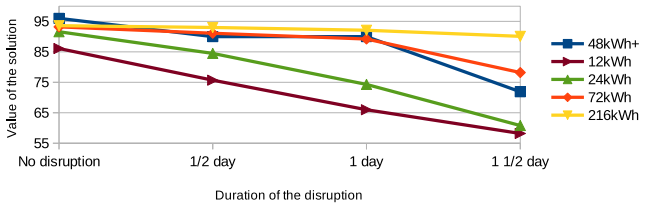

Fig. 3. Impact of the battery size of the taxis on the solution, during a disruption. There are 2 fast chargers, except for '12kWh+' where there are 3. The disruption finishes Friday at Midnight.

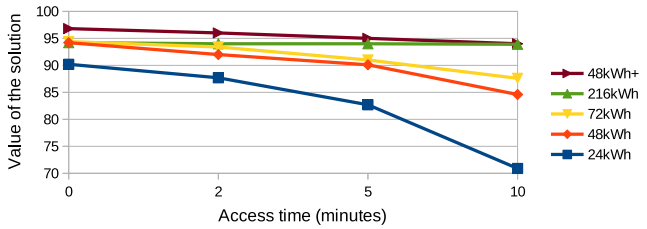

Fig. 4. Impact of the battery size of the taxis on the solution, depending on the time needed to access to a charger (queue + docking / undocking time). There are 2 fast chargers, except for '48kWh+' where there are 3.

D. Parking Search & Charging Queue

With 2 fast chargers and 72kWh batteries, taxis finding a parking place immediately or having to search for one hour does not have a significant impact on the value of the solution. However, the energy spent changes very significantly: 234kWh spent per taxis during the simulation with no parking search, but 311kWh spent with a one hour search needed. The usage rate of the chargers goes from 58% to 77%. However, if there is only one charger, then it is used constantly. In this situation, the value of the solution is 78.6% if parking places are found instantly, 73.2% with 15-minutes searches, and 72.1% with one-hour searches. Overall, the value of the solution starts to drop sharply once the use rate of the chargers goes above 80%.

We evaluated the impact of battery size on the value of the solution depending on the amount of time taxis have to wait at a charger before a charge can start (see figure 4). The most interesting result is the comparison between "48kWh" and "48kWh+". A charging infrastructure shared with other users, such that 10 minutes of queue are consistently needed to access to any charger is better than a charging infrastructure only useable for the fleet, but 33% smaller. The result remains the same for any taxi battery size. This conclusion is reversed however if either chargers are slower than 24kW, or if the number of chargers is doubled.

E. Scalability

CPLEX without tuning was used on small instances to test its limits. With 2 taxis and 2 chargers useable twice each, CPLEX needed more than 2 hours to find a solution when the number of requests was above 17. This prevented us from comparing our method to instances where the best solution is known and not trivial, where multiple partial charges are needed, while remaining somewhat comparable to realistic instances.

Our greedy needed less than one minute for every instance. Reducing the size of the taxi battery increased significantly the number of charges needed, and since the charge insertion has a quadratic complexity, a significant rise in computation time was observed.

Without surprises, the simulated annealing worked better on smaller instances, with the same computation time, allowing for improvement above +10% for instances with less than 4000 requests. Increasing computation time did not allow for much more improvement however. An analysis of the schedules of the chargers showed that the simulated annealing has a tendency to fragment charges, which increases the number of deadhead trips. Neighbour moves which modify charges directly could solve this problem. Only moving requests in a way which may remove/shorten charges does not seem to be sufficient.

V. Conclusion

The transition from ICE vehicles to electric cars cannot be done without consideration for the broad difference in performance of these technologies, in terms of vehicle range as well as refilling speed. These considerations matter especially to autonomous taxi fleet managers, whose vehicles are used heavily. To tackle this problem, we developed scalable heuristics which take into account the possible limitations of a charging infrastructure.

In our greedy heuristic, inserting the most valuable requests first seems to give good results. The best variant of this greedy might consists in inserting requests with the highest value while being the least constraining. However, determining if a request is constraining or not is not trivial, particularly when considering charging and ride sharing. A simulated annealing can improve this solution. In our annealing, charges are only inserted in reaction to a need, after a request insertion. Having a move in the simulated annealing specifically designed to reorganise charges should be considered.

In normal circumstances, our heuristic does not benefit much from large batteries. However, large batteries are still useful to handle unusual situations. Taxis with 72kWh seem able to handle 1-day disruption. Large batteries also fare better when taxis need to queue before every charge. We compared private charging network to a shared one, in regard to their efficiency. A private network is more efficient when the network is far from saturated, or when taxis have a very small battery. A slightly bigger but shared network is better otherwise. Many other aspects have to be considered, however, since this choice entails many differences.

Whether parking search matters depends a lot on the use of the charging infrastructure. Once it get saturated,

increased search time significantly decreases the solution, but it does not have a sensible effect before. A limitation of our objective function is that it does not take into consideration the energy spent. In some instance, parking search accounted for more than 50% of the energy spent. More data on parking search would be useful to make more realistic simulations.

References

[1] P. M. Bösch, F. Becker, H. Becker, and K. W. Axhausen, "Cost-based analysis of autonomous mobility services," Transport Policy, vol. 64, pp. 76 – 91, 2018.

[2] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte, "Static pickup and delivery problems: A classification scheme and survey," in TOP: An Official Journal of the Spanish Society of Statistics and Operations Research, vol. 15, 02 2007, pp. 1–31.

[3] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem (darp): Models and algorithms," in Annals OR, vol. 153, 06 2007, pp. 29–46.

[4] S. C. Ho, W. Szeto, Y.-H. Kuo, J. M. Leung, M. Petering, and T. W. Tou, "A survey of dial-a-ride problems: Literature review and recent developments," Transportation Research Part B: Methodological, vol. 111, pp. 395 – 421, 2018.

[5] S. Erdoğan and E. Miller-Hooks, "A green vehicle routing problem," Transportation Research Part E: Logistics and Transportation Review, vol. 109, p. 100–114, 01 2012.

[6] M. Schneider, A. Stenger, and D. Goeke, "The electric vehicle-routing problem with time windows and recharging stations," Transportation Science, vol. 48, no. 4, pp. 500–520, 2014.

[7] Ángel Felipe, M. T. Ortuño, G. Righini, and G. Tirado, "A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges," Transportation Research Part E: Logistics and Transportation Review, vol. 71, pp. 111 – 128, 2014.

[8] D. Goeke and M. Schneider, "Routing a mixed fleet of electric and conventional vehicles," European Journal of Operational Research, vol. 245, no. 1, pp. 81 – 99, 2015.

[9] G. Hiermann, J. Puchinger, S. Ropke, and R. F. Hartl, "The electric fleet size and mix vehicle routing problem with time windows and recharging stations," European Journal of Operational Research, vol. 252, no. 3, pp. 995 – 1018, 2016.

[10] A. Montoya, C. Guéret, J. E. Mendoza, and J. G. Villegas, "The electric vehicle routing problem with nonlinear charging function," Transportation Research Part B: Methodological, vol. 103, pp. 87 – 110, 2017, green Urban Transportation.

[11] C. Bongiovanni, M. Kaspi, and N. Geroliminis, "The electric autonomous dial-a-ride problem," Transportation Research Part B: Methodological, vol. 122, pp. 436–456, 04 2019.

[12] M. Chester, A. Horvath, and S. Madanat, "Parking infrastructure: energy, emissions, and automobile life-cycle environmental accounting," Environmental Research Letters, vol. 5, no. 3, p. 034001, 2010.

[13] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," Oper. Res., vol. 35, no. 2, pp. 254–265, Apr. 1987.

[14] B. Roberto, E. Bartolini, A. Mingozzi, and R. Roberti, "An exact solution framework for a broad class of vehicle routing problems," Computational Management Science, vol. 7, pp. 229 – 268, 2010.