Transportation Research Part C 161 (2024) 104551

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

A filtering system to solve the large-scale shared autonomous vehicles Dial-a-Ride Problem

Chijia Liu [a,*], Alain Quilliot [a], Hélène Toussaint [a], Dominique Feillet [b]

[a] Université Clermont Auvergne, CNRS, UMR 6158 LIMOS, 63178 Aubière, France
[b] Mines Saint-Etienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, F - 42023 Saint-Etienne, France

ARTICLE INFO

Keywords:
Shared autonomous vehicles
Dial-a-Ride Problem
Ride-sharing
Transit-network graph
Large-scale vehicle routing

ABSTRACT

Shared autonomous vehicles are rapidly becoming one of the most popular topics in the fields of transportation and operations research. This paper studies a prospective transportation system in which shared autonomous vehicles (SAVs) providing Dial-a-Ride services in urban and rural areas must meet a large number of passenger requests (at least 10^5 per day). We consider those SAVs likely to replace, in the future, most of the individual vehicles inside urban areas. We address this very large-scale problem with a greedy insertion heuristic that involves a specific filtering system design in order to speed up the decision process and relies on an original *network-driven* encoding of the routes. Experimental results show that en-route vehicles can be drastically reduced by such systems using a fleet of SAVs of capacity 10, decreasing the vehicle number by more than 98%, compared to a situation where all travel demands would rely on personal vehicles. In addition, thanks to the filtering system, the total execution time can be reduced by almost 97% compared to the classic best-fit insertion heuristic, while maintaining the quality of the solution. This opens the way to the adaptation of our algorithms to a dynamic real-time context.

1. Introduction

With the rapid progress of technology and the continuous improvement of quality of life, more and more private vehicles are currently overcrowding our urban areas. However, while private cars provide convenience to human beings, they also induce pollution due to CO^2 emissions and traffic pressure, especially during peak hours in urban districts. A consequence is that governments are now encouraging people to use public transport and promoting new transportation systems, services, and technologies.

Two categories of such new services are becoming predominant: vehicle-sharing and ride-sharing. Vehicle-sharing systems (Gavalas et al., 2015; Nansubuga and Kowalkowski, 2021), such as Zipcar and Car2Go, provide users with short-term access to vehicles, offering an alternative to traditional vehicle ownership. Such systems not only facilitate convenient access to vehicles without the commitment of ownership, but also contribute to reducing the environmental impact linked with the widespread private vehicle ownership.

Ride-sharing is a service that enables people to share a common journey in the same vehicle. Such a system, like Uber Pool, Lyft Shared Rides, and Didi Hitch, aims to avoid too many almost empty vehicles running on the streets and thereby reduce the number of effectively used private vehicles (Agatz et al., 2012; Furuhata et al., 2013). It is worth noting that in some literature, this kind of service is also interchangeably termed "ride-pooling". For clarity within this paper, we will stick to the term "ride-sharing". These

* Corresponding author.
E-mail addresses: chijia.liu@uca.fr (C. Liu), alain.quilliot@uca.fr (A. Quilliot), helene.toussaint@uca.fr (H. Toussaint), feillet@emse.fr (D. Feillet).

https://doi.org/10.1016/j.trc.2024.104551
Received 11 July 2023; Received in revised form 25 February 2024; Accepted 29 February 2024
Available online 10 March 2024

two categories may merge into hybrid systems where vehicles are owned by operators and left for access by users. For instance, in a dial-a-ride (DAR) system, the operator plays the role of *master of the game*: It routes and schedules the vehicles in order to meet the demands of the passengers. DAR services were initially dedicated to elderly and disabled people. Nowadays, DAR services are more and more implemented as demand-responsive transportation services in urban, peri-urban, or even rural areas (Ho et al., 2018). An obstacle to their extension as a mass transportation tool is the cost related to the drivers.

However, the current emergence of autonomous vehicle (AVs) technologies, one of the rising topics in the field of transportation and operational research (Rossi et al., 2018; Bongiovanni et al., 2019; Alonso-Mora et al., 2017), may provide us with an answer to this cost issue. Along with the rapid immersion of the sharing economy, Shared Autonomous Vehicles (SAVs) have been increasingly mentioned in the literature over the past few years (Levin et al., 2017; Narayanan et al., 2020; Merat et al., 2017). It seems more and more plausible that SAV might be the tool for turning DAR services into flexible mass transportation systems.

In this paper, we consider such SAV-based DAR systems likely to replace in the future an important part of individual vehicles inside urban areas. So we study a prospective DAR system, involving medium-capacity SAVs and designed to meet a very large number of passenger requests (more than 10^5). The adoption of such a system today is still dampened by many factors: the link that people maintain with their vehicles, their reluctance to share vehicles, the limits of AVs technologies, and the strength of habits... (Maghraoui et al., 2020). However, the necessity to alleviate the traffic pressure caused by individual vehicles and reduce carbon emissions, together with concerns about the price of energy, suggests that the existence of such an innovative system might become more and more plausible in the next few decades, significantly reducing the use of individual motorized transportation media in the urban landscape. So our objective here is to provide an insight into the operational management of such a system and its ability to reduce traffic on an urban transit network.

This leads us to address a large-scale static Dial-A-Ride Problem (DARP) involving SAVs, while focusing on the global costs induced by the vehicles. In view of the high number of passenger requests, we solve it with a greedy insertion algorithm that is adaptable to dynamic contexts. Then, the underlying key issue becomes the design of the insertion procedure: how shall we proceed in order to deal efficiently with the insertion of requests once several hundred vehicles are currently running with thousands of passengers on board? According to this, our main contributions are as follows:

- We introduce a *filtering system* in order to quickly identify good insertion parameters every time we deal with the insertion of a request. This system contains:
    1. Specific *filtering data structures* based upon a clustering of the transit network and of the time space into zones and periods, that provides us with a fast access to the vehicles likely to meet a given new request, as well as to related insertion parameters;
    2. Specific *filtering procedures* that compute from those filtering data structures, every time we deal with a given target request, the best-fitting vehicles and the corresponding insertion positions;
    3. A *stopping mechanism* based upon a notion of *computational effort* and the introduction of an *effort counter* φ and an *effort* threshold ϕ, together with a *regeneration mechanism* aiming to cope with failures.
- This system relies on a specific *network-driven* encoding of the routes followed by the vehicles, that fits with our filtering data and procedures, and that speeds the feasibility tests required by any insertion.
- We show how our greedy heuristic, designed in order to provide an estimation of the number of vehicles likely to be involved into our DAR system, may be adapted to operational dynamic contexts.
- We provide computational results that show on a representative real network how the large-scale ride-sharing system can affect the flow of vehicles circulating in the transit network. Those results show that our SAV DAR system would induce a reduction of approximately 98% in the number of vehicles required using a fleet of SAVs of capacity 10, compared to the scenario where each request would be served by individual vehicles. Such a reduction in the fleet size would cause a slight increase in the time that passengers spend inside the vehicles due to the necessary detours made by SAVs, but it would also induce a significant decrease of the traffic, even during peak hours. Furthermore, our filtering system would reduce the running time of the decision process by almost 97%, without any significant loss in the quality of the solution. This opens the way to an efficient integration of this decision process into operational dynamic contexts.

The structure of the paper is as follows. In Section 2 we propose a state-of-the-art on related works. In Section 3, we formally define our problem and motivate underlying assumptions. Then, in Section 4 we describe our generic algorithmic framework *ProcessDARP*, together with our *network-driven* representation of the routes and related insertion procedures, and discuss our *spatial–temporal* insertion strategies. In Section 5, we focus on the *Search_Insert* sub-problem, which is about the search for the best parameters for the insertion of a target request, and describe the algorithmic scheme that we use to handle this sub-problem, before entering in Section 6 into the details of our *Filtering System*. Section 7 discusses the way our *ProcessDARP* algorithm might be adapted to dynamic contexts and Section 8 presents experimental results. We conclude in Section 9.

2. Literature review

This paper investigates a promising large-scale SAV DAR system within on-demand mobility transportation. Therefore, this section begins by examining recent studies on large-scale on-demand mobility systems before focusing on specific models and algorithms tailored for the Dial-A-Ride Problem (DARP).

2.1. Large-scale on-demand mobility systems

On-demand mobility transportation systems offer customers personalized and efficient transportation services. Such systems leverage digital platforms and shared mobility concepts to provide convenient, flexible, and cost-effective travel options. As the demand for efficient urban transportation grows, studies of large-scale on-demand mobility transportation systems have become more prevalent. Studies take interest in analyzing different characteristics of the systems, such as the management of congestion problems caused by the large size of the vehicle fleet (Levin, 2017; Liang et al., 2020), the recharging issue if vehicles are electric powered (Bongiovanni et al., 2019), and the potential and benefit if ride-sharing can be introduced to such a system. For example, in Fagnant and Kockelman (2018), the authors study a dynamic Ride-Sharing System (DRS) of SAVs in Austin, Texas, based on simulations under different scenarios. Results suggest that, within a 24 mi × 12 mi geofence with 56,324 requests, when DRS parameters are loosened, greater ride-sharing may greatly reduce net vehicle miles traveled and passengers' waiting times, especially during peak hour, and improve passengers' total service time. As one of the characteristics of our SAV DAR system, we also include the possibility of ride-sharing, meaning that different requests are able to share some part of their journey in the same vehicle.

In a customer-facing system like the one we are studying in this work, users' demands should be processed rapidly, especially in dynamic situations where new requests keep entering the system in real-time. Consequently, it is the key to accelerating the routing and scheduling process.

A first approach consists of working on the data either through some preprocessing or at the time the decision is taken. Different techniques, such as reinforcement learning and deep learning, can be applied to road networks and passenger requests to mine data more comprehensively. In Nazari et al. (2018), the authors introduce a framework that integrates reinforcement learning to solve a wide range of combinatorial optimization problems. They have shown that their framework is capable of solving VRPs and that their approach outperforms state-of-the-art VRP heuristics. In Kullman et al. (2021), Kullman et al. have considered an Electric Ridehail Problem with Centralized Control (E-RPC) where a centrally controlled EV fleet is used to provide ride-hail services. In addition, the operator also envisages the EVs' relocation and recharging issues. In their paper, they propose policies derived from deep reinforcement learning to solve the E-RPC. They demonstrate that their solution significantly outperforms the benchmark reoptimization policy based on test results with real operation data from the island of Manhattan in New York City. In Riley et al. (2020), the authors study a large-scale real-time ride-sharing system. They train a Vector Autoregression(VAR) model with historical requests to predict the demands for a geographical zone z at time t. Their approach uses prediction to relocalize idle vehicles, which also naturally helps retrieve the nearest vehicles for target requests.

Another approach is to introduce filters based on different criteria to avoid the time-consuming planning and scheduling process. For example, in Bertsimas et al. (2019), the authors study a large-scale on-demand taxi system. In order to leverage the structure of real-world vehicle routing applications to make mixed-integer formulations tractable, they hierarchically integrate different approaches into a basic Mixed Integer Programming (MIP) model. Typically, several filters that decrease the number of binary decision variables are introduced based on the feasibility of the vehicle's move from one location to another. Under a re-optimization scheme, the resulting algorithm is able to dispatch in real time thousands of taxis serving more than 25,000 customers within a horizon of 1.5 h, while providing better routing solutions compared to other state-of-the-art heuristics. In Shuo Ma et al. (2013), the authors propose a large-scale taxi ride-sharing system called *T-Share*, where users' demands are submitted in real time. Every demand is assigned a pickup and delivery window. Every time a new demand enters the system, the operator should assign it to a taxi while scheduling the taxi's route. The system is supposed to serve at least 330,000 requests per day with a fleet of 2980 taxis. To accelerate the dispatching process, the authors partition the road network into zones and introduce a spatial–temporal index to rapidly retrieve a subset of interesting (nearest) vehicles that are regarded as candidates to serve the given request. They propose two algorithms: *single-side taxi searching* screens from the request's origin side, while *dual-side taxi searching* takes both the origin and destination sides into account. The experiments were carried out using the real network of Beijing, which contains 106,579 road nodes and 141,380 road segments. The results showed that *dual-side taxi searching* is 50% more efficient than the *single-side taxi searching* algorithm, while the increase in travel distance could be neglected. Similar to Shuo Ma et al. (2013), in our work, we introduce a filtering system that greatly relies on spatial–temporal index matrices. Furthermore, our filtering system is not only able to quickly retrieve interesting candidate vehicles to serve a target request r but also plausible insertion positions within the schedules of each candidate vehicle.

2.2. Dial-A-Ride Problem

Within the context of an SAV DAR transportation system, our reliance naturally gravitates towards the Dial-A-Ride Problem (DARP). Despite this, there is a scarcity of pertinent studies on on-demand mobility systems specifically addressing DARP. As our system is built upon the framework of the DARP, it is crucial to dig into existing studies focused on this problem.

The DARP consists of designing the routes and schedules of a fleet of vehicles so as to serve a set of passenger requests, typically characterized by a pickup location, a drop-off location, pickup and/or drop-off time windows, and a maximal ride time. Decisions are driven by objectives such as minimizing vehicle operating costs or maximizing customer service quality. DARP is NP-hard as it admits the Vehicle Routing Problem (VRP) as a special case. So, very few studies seek to solve the DARP using exact optimization methods, unless for solving small-sized static problems (Cordeau, 2006; Rist and Forbes, 2021; Schulz and Pfeiffer, 2024). More studies tend to propose approximate methods such as heuristic or meta-heuristic algorithms, which are capable of solving larger instances (Gendreau and Tarantilis, 2010; Molenbruch et al., 2017; Ho et al., 2018). In Cordeau and Laporte (2003), the authors propose a tabu search meta-heuristic to solve a DARP that considers constraints such as pickup and drop-off time windows of requests, longest vehicle

detour, and maximum passengers' ride time. In Calvo and Colorni (2007), authors introduce an effective heuristic that implements a two-phase scheduling scheme. The strategy consists of classifying all submitted requests into different clusters in the first place and solving the single-vehicle DARP for each cluster. In the second phase, swaps between clusters are performed to obtain better feasible solutions. Another commonly employed method for addressing the DARP is the Large Neighborhood Search (LNS), along with its extension known as the Adaptive Large Neighborhood Search (ALNS). In Jain and Van Hentenryck (2011), an ALNS heuristic is presented that repeatedly uses constraint programming to find good reinsertions of requests. Competitive results have been reported on instances with the number of requests ranging from 24 to 144.

When it comes to dynamic DARP, the problem size is usually much larger, aligning more closely with the complexities observed in real-life scenarios. The size of the problem, and more specifically the number of requests, becomes an obstacle to setting a mathematical programming formulation (*MIP* model, etc.), and it is difficult to rely on a local search (*Build and Destroy*, *Large Neighborhood Search*, etc.) approach or on a tree search (*Branch and Bound*, etc.) approach. In this case, the greedy insertion heuristics are one of the most popular methods to address the problem due to its simplicity and effectiveness (Diana and Dessouky, 2004; Tafreshian et al., 2021). Considering the very large-scale aspect of the problem, our algorithm is also based on a best-fit greedy insertion heuristic. In addition, although we mainly focus on the static version of the problem, we want an algorithmic framework adaptable to a dynamic context. A greedy insertion decision process is likely to be the best tool to fill the gap between static and dynamic. Insertion heuristics necessitate addressing two key questions: determining which request to insert at each iteration (i.e., insertion order) and deciding where to insert it within all the routes (i.e., insertion parameters).

Regarding the insertion order, a prevalent approach involves inserting requests in chronological order based on their departure times. To the best of our knowledge, we are the first to introduce an insertion order strategy that considers the geometric similarity of requests.

Regarding the insertion parameters, efficient insertion feasibility tests are crucial. In Hunsaker and Savelsbergh (2002), an algorithm aimed at efficiently verifying insertion feasibility within a route is discussed. This algorithm handles pickup and drop-off services with associated time windows, service duration, and maximum ride time for requests. Initially claimed to operate in $O(n)$ complexity, Tang et al. (2010) later identified a flaw and proposed a refined algorithm with a time complexity of $O(n^2)$. Likewise, in another study (Varone and Janilionis, 2014), a sequential approach integrated into a best-fit insertion heuristic is employed. Their approach involves the calculation and combination of four sets of shortest paths regarding the target request, and then potential vehicles and their feasible routes are gradually revealed while ensuring the maximum ride time constraint is met. In Shuo Ma et al. (2013), the insertion feasibility check for requests in vehicles is optimized using a *Lazy Shortest Path Calculation Strategy*. This strategy estimates travel times between locations using pre-computed inter-cell travel times and the triangle inequality. If the estimated travel time does not meet feasibility criteria, the insertion position is rejected. This approach notably reduces computational costs by 83%. Alternatively, some authors suggest disregarding certain insertion positions (Horn, 2002). In our work, in addition to the filtering system that efficiently selects valuable insertion parameters, we also provide an original *transit network-based* route modeling approach that minimizes the route length and significantly accelerates the insertion feasibility tests in a large-scale, high-mutualization context.

3. Problem statement

We now consider a prospective SAV-based transportation system that offers DAR services in the urban and rural areas of a city. We define this system following the hypotheses listed below:

- The underlying transit network contains only one depot. The time space during which our DAR system runs is denoted as $[t_0, T]$, where t_0 is the beginning time of the services (which is in practice set as $t_0 = 0$), and T means the *time horizon*. This simplified hypothesis can be generalized to a multi-depot setting where each SAV is attached to a depot. All SAVs are assumed to be identical with the same capacity Q.
- Passenger requests involve a pickup time window and a maximum ride time. They are non-preemptive, which necessitates that a single vehicle fulfill each request only once.
- We neglect the pickup and drop-off service times so that passengers get on board as soon as the vehicle shows up, and they are dropped off immediately upon arriving at their destination.
- We impose that no request can be rejected, which means that one of our concerns is about the size of the fleet and that this size becomes a key component of our problem.
- SAVs are assumed to be electric-powered. Nevertheless, due to the very fast evolution of technologies related to batteries (autonomy, recharge speeds, etc.), we decide not to take care here on the recharging issue.

The related **LS_DARP:** ***Large Scale DARP*** problem is about the way we route and schedule the vehicles and the way we assign requests to the vehicles so that both the number of active vehicles and the running costs of the vehicles are minimized. The QoS (*Quality of Service*) criterion is supposed to be included in the time window and the maximal riding time constraints associated with each request. Before setting it in a more formal way, we first discuss the two following key features of our **LS_DARP** problem:

1. We deal with a large network (from 1000 to 15,000 nodes) that tends to make appear a large set of possible request pickup and drop-off locations inside a medium-sized urban/peri-urban area and with a large number of requests (from 10,000 to 300,000). This corresponds to our interpretation of such an SAV DAR system as capturing a significant part of the daily commutes

in the service area. In the case of a medium-sized area like Clermont-Ferrand (with an active population of 100,000),[1][2] it is estimated that at least 300,000 commutes should be made during weekdays.[3] Consequently, the problem sizes under consideration depict scenarios ranging from moderate to exceedingly high utilization of the SAV DAR system.

2. Our initial setting of the **LS_DARP** problem is static, which means that all requests are known in advance and no decision is taken online. This fits well with strategic concerns about the size of the fleet and the operational costs supported by the SAV operator. However, in Section 7, we shall discuss how our approach might allow us to shift towards a dynamic setting.

Remark 1. As mentioned previously, we motivate the reference to autonomous vehicles by the fact that they do not induce any driver cost. So they might provide the kind of flexibility wanted by people who currently rely on their own vehicles without the high fares demanded by taxi-like systems. Clearly, managing a fleet of autonomous vehicles may require addressing some specific issues and taking into account additional criteria, namely safety and reliability. But as in the case of the recharging issue, the uncertainty about the fast evolution of SAV technologies leads us to not take care of those specific issues.

Let us now set up our **LS_DARP** in a formal way.

We consider a transit network denoted by $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. The node set $\mathcal{N}$ represents the potential SAV pickup and drop-off locations, and the arc set $\mathcal{A}$ represents the elementary moves that the SAV may perform. The depot is a node in $\mathcal{N}$, denoted by n_0. For any arc $a_{ij} \in \mathcal{A}$, where $i, j \in \mathcal{N}^2$, $t(a_{ij})$ represents its traversing time. By extension, given any node pair $u, v \in \mathcal{N}^2$, we utilize $t(u, v)$ to denote the time it takes for an SAV to move from node u to node v along the shortest (fastest) path.

We suppose that the shortest path distance values have been pre-computed as part of a pre-process, and that we shall be able, all throughout the execution of our algorithms, to get direct access to them. Note that such an assumption implies that we do not cope here with time dependence.

We consider a set $\mathcal{R}$ of passenger requests. A request $r \in \mathcal{R}$ involves a number of passengers q^r, a pickup service P^r and a drop-off service D^r: P^r includes a pickup location $p^r \in \mathcal{N}$ and an associated time window $[e_{P^r}, l_{P^r}]$, with e_{P^r} the earliest pickup time for r, and l_{P^r} the latest timestamp for r to be in the vehicle; D^r includes a drop-off location $d^r \in \mathcal{N}$ and a *maximum ride time* $T^r \geq t(p^r, d^r)$. In our experiments, we shall set, for any request r:

$$T^r = \alpha \cdot t(p^r, d^r),$$

where $\alpha > 1$ is a control parameter independent of r. In addition, we derive, for every request $r \in \mathcal{R}$, a drop-off time window $[e_{D^r}, l_{D^r}]$ with $e_{D^r} = e_{P^r} + t(p^r, d^r)$ and $l_{D^r} = l_{P^r} + T^r$.

In Section 7, regarding the adaptation of our algorithms to dynamic **LS_DARP** contexts, every request r will also be associated with an *emission time value* $t^r_{emit} < e_{P^r}$, meaning the time when r was emitted.

Requests are carried out thanks to an *a priori* unlimited fleet of identical SAVs. The capacity of these SAVs is denoted by Q. Vehicle routes must start from the depot, finish at the depot, and take place inside a time space $[t_0, T]$, the time horizon T being at least equal to any value l_{D^r}. The load of a vehicle v may never exceed the vehicle capacity. The route and the schedule followed by a vehicle v must be such that, for every request r assigned to v:

- The pickup service P^r precedes the drop-off service D^r;
- The schedule of v (the departure and arrival times associated with the nodes of its route) is consistent with both the pickup window of r and its maximal ride time.

As for the objective function, we assume that a "large-scale" mass-oriented SAV fleet should be able to serve almost all the requests. This makes the size of this fleet a key issue. Besides, SAVs being assumed to be electric-powered, while their subsequent operational expenses, such as maintenance and repair costs, are generally much lower compared to those of traditional Internal Combustion Engine vehicles, the upfront purchase costs are typically higher. Therefore, our primary objective is to use as few SAVs as possible. Secondly, for the sake of energy savings, the total drive time of the SAVs is considered a secondary criterion to measure the quality of the decision. This leads to a definition of our objective function as a lexicographic combination of two criteria: Given two feasible solutions, Θ_1 with a final fleet size V_1 and a total drive time T_1 and Θ_2 with a final fleet size V_2 and a total drive time T_2, we say that Θ_1 is better than Θ_2 if and only if $V_1 < V_2$, or $V_1 = V_2$ and $T_1 < T_2$. As for the QoS criterion, we consider it implicitly included in the constraints induced by the pickup time windows and the maximal ride times.

The notation is summarized in Table 1.

Remark 2. As mentioned earlier, we do not address here the time dependence issue and this allows us to pre-compute shortest path values $t(u, v)$ in order to handle them in direct access while running our algorithms, even though the number of nodes of $\mathcal{G}$ may be large. However, we firmly believe that an effective DAR system should be capable of accommodating time-dependent events. Our algorithmic approaches possess the requisite flexibility for adaptation to such scenarios: we adopt a strategy of postponing the utilization of travel times $t(\cdot, \cdot)$ to the latest possible stage, which mitigates the need for computationally expensive, upfront calculations of travel times when transitioning to time-dependent contexts.

[1] Insee, Complete File Clermont-Ferrand 2023 https://www.insee.fr/fr/statistiques/2011101?geo=COM-63113.

[2] Insee, Department of Puy-de-Dôme: Population and employment are polarizing around Clermont-Ferrand, https://www.insee.fr/fr/statistiques/5209717?sommaire=5213119&q=puy+de+dome.

[3] SDES, https://www.statistiques.developpement-durable.gouv.fr/comment-les-francais-se-deplacent-ils-en-2019-resultats-de-lenquete-mobilite-des-personnes.

Table 1
Notation (LS_DARP definition).

Notation	Definition
$\mathcal{G} = (\mathcal{N}, \mathcal{A})$	Road-network graph (with GPS coordinates)
n_0	Depot ($n_0 \in \mathcal{N}$)
$t(u, v)$	Travel time between nodes u and v ($u, v \in \mathcal{N}$)
$\mathcal{R}$	Set of passenger requests
P^r	Pickup service ($r \in \mathcal{R}$)
D^r	Drop-off service ($r \in \mathcal{R}$)
p^r	Pickup location ($r \in \mathcal{R}$)
d^r	Drop-off location ($r \in \mathcal{R}$)
q^r	Number of passengers ($r \in \mathcal{R}$)
$[e_{P^r}, l_{P^r}]$	Pickup time window ($r \in \mathcal{R}$)
$[e_{D^r}, l_{D^r}]$	Drop-off time window ($r \in \mathcal{R}$)
T^r	Maximum ride time ($r \in \mathcal{R}$)
Q	Vehicle capacity
t_0	Beginning time of all DAR services
T	Time horizon

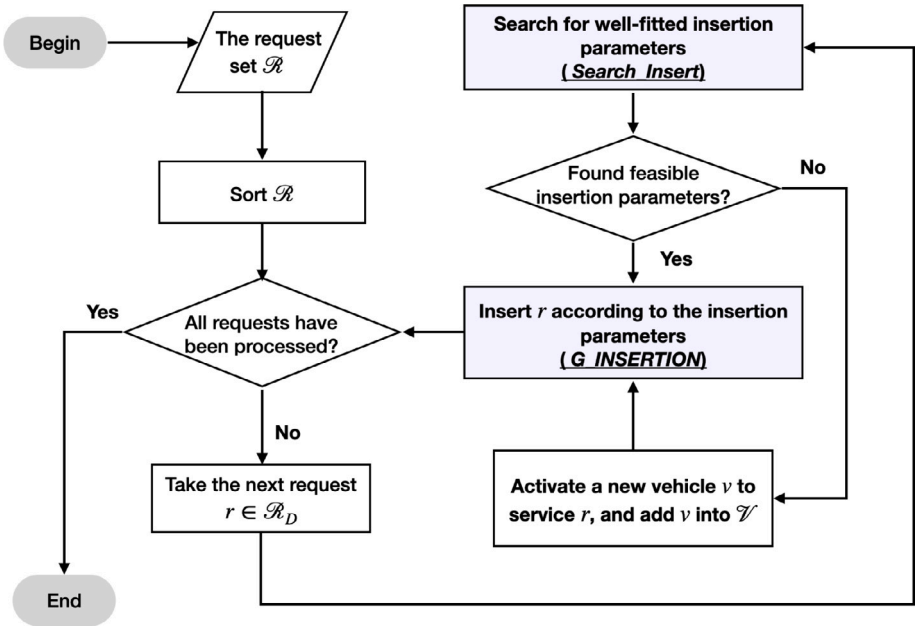


Fig. 1. The main algorithmic framework: *ProcessDARP*.

4. Generic algorithmic framework

In this section, we introduce our algorithmic framework *ProcessDARP* to deal with **LS_DARP**, which is based on a standard best-fit greedy insertion heuristic (see Fig. 1):

- Requests are sequentially processed according to a specific order until all of them have been inserted.
- When we receive a request r, we are provided with a current fleet $\mathcal{V}$ of *active* vehicles and with the collection $\Theta = \{\theta^v, v \in \mathcal{V}\}$ of *routes* followed by the vehicles. Every route θ^v is given together with the schedule (request drop-off and pickup times) of the actions performed by v while running along θ^v.
- Then, given such a target request r we apply a *Search_Insert* procedure that searches for a vehicle $v \in \mathcal{V}$ together with some feasible *insertion parameters*, such that the induced insertion cost is minimal. If this procedure succeeds, we apply a *G_INSERTION* procedure that effectively performs the insertion of r according to the corresponding insertion parameters in the route of v, and updates the related route θ^v. Otherwise, we activate a new vehicle v and add it to $\mathcal{V}$, together with a trivial route reduced to the pickup and drop-off locations of r.

In our large-scale context, efficiently implementing this generic *ProcessDARP* framework requires discussing the following points:

- **The representation of the routes θ^v:** Standard approaches rely on a virtual complete *request-based* graph, which is made of all the pickup and drop-off locations of the requests, and consider a route as a path in this graph. In our case, this approach would

lead to very large routes and deprive us of any means to link the routes with topological patterns. Therefore, we propose in Section 4.1 a *transit network-based* modeling of the vehicle routes, and adapt the insertion mechanisms to this representation.

- **The insertion order strategy**: The *insertion ordering* according to which we manage the requests may significantly impact the quality of the *ProcessDARP*. We discuss in Section 4.2 different ways this ordering may be defined, while taking into account both their temporal and geographical features.
- **The search for well-fitted insertion parameters**: Finally, given a request r, our main issue is the design of the *Search_Insert* procedure. This procedure consists of selecting the target vehicle v and the related insertion parameters, which are *positions* for both the pickup and drop-off services of r in the route θ^v. In standard DARP, one tries all the vehicles and scans every route in order to identify the best-fitted insertion parameters. However, the very large number of requests and the concern of adapting our algorithm to a real-time context lead us to refine this approach. Specifically, we introduce a filtering system into the *Search_Insert* process, which takes advantage of our transit network-based route modeling to search for valuable insertion parameters regarding a given request r and a current route collection Θ. We devote Section 5 to describing the general resolution scheme of *Search_Insert*, then provide in Section 6 the details of the related filtering system.

4.1. Routes and insertions

A solution to **LS_DARP** is a collection $\Theta = (\theta^v, v \in \mathcal{V})$, where θ^v is the *route* followed by vehicle v, and $\mathcal{V}$ contains all the SAVs used during the DAR process. As mentioned previously, most studies on DARP define a route as a sequence of pickup/drop-off identifiers p^r, d^r, with $r \in \mathcal{R}$, together with time windows. In our large-scale context, such an approach would have several inconveniences: Firstly, it is likely to result in very long routes: Assuming 2000 vehicles to meet 300,000 requests, a route modeled this way contains in average 300 nodes. In the worst case, searching for best-fitted insertion locations for P^r and D^r in θ^v would require a computational effort of $O(N(v)^2)$, where $N(v)$ is the number of nodes in θ^v. Moreover, in DARP with time window and maximum ride time constraints, the constraint propagation process should be invoked when testing the insertion feasibility of a request r into a route θ^v. Depending on the implementation, the complexity is in general $O(N(v)^2)$ (Hunsaker and Savelsbergh, 2002; Tang et al., 2010). Besides, such a *request-based* route modeling makes it difficult to capture the geographical patterns of routes and the mutualization of services within a route, as we do not have the direct link between a route and the transit network $\mathcal{G}$.

In this study, we adopt a *transit network-based* route modeling. Rather than distinguishing the pickup and drop-off services of each request, all services that are to be operated consecutively at the same location in a route are merged into what we call a *key point*. In this way, a vehicle route is represented as a sequence of key points. Fundamentally, a key point is a node n of the network $\mathcal{G}$ together with a *package* of services related to n. We formalize this by defining a key point K as a record of the following data:

(a) $n_K \in \mathcal{N}$: the *physical* node of $\mathcal{G}$ related to K, at which vehicle v will effectively perform at least one pickup or drop-off service. For the sake of simplicity, from now on for two key points $K \in \theta^v$ and $K' \in \theta^v$, we use $t(K, K')$ to denote the travel time from n_K to $n_{K'}$;
(b) R_K^+: a list of requests arranged to get onboard at K. We term these requests as *inbound* requests of K;
(c) R_K^-: a list of requests arranged to get off the vehicle at K. We term these requests as *outbound* requests of K;
(d) $q_K \in \mathbb{N}$: the number of passengers on board just before leaving K;
(e) $[e_K^a, l_K^a]$: the arrival time window at K, where e_K^a (resp. l_K^a) represents the earliest (resp. latest) time to arrive at K.
(f) $[e_K^d, l_K^d]$: the departure time window of K, where e_K^d (resp. l_K^d) represents the earliest (resp. latest) time to leave K. Note that the arrival and departure time windows differ, because the vehicle should wait until all the inbound requests are onboard before leaving.

For a key point $K \in \theta^v$, we denote by $Pred(K)$ its predecessor in θ^v, i.e., the previous key point scheduled to be visited before K, and by $Succ(K)$ its successor in θ^v, i.e., the next key point scheduled to be visited after K.

Let us now describe how to adapt the notion of *insertion* into θ^v regarding our route modeling (Section 4.1.1) as well as how to update arrival and departure time windows every time we effectively perform an insertion (Section 4.1.2).

4.1.1. Insertion procedures

Inserting a request r into a route θ^v means first inserting its pickup service P^r then the drop-off service D^r. Let us denote the key points supporting the services P^r and D^r as, respectively, $K(P^r)$ and $K(D^r)$. Now, for a service $X \in \{P^r, D^r\}$, we define the following two insertion types at an insertion position K:

- *Regular Insertion*
  For a regular insertion, we consider inserting X *between key points*. As illustrated in Fig. 2, a regular insertion consists in inserting $K(X)$ between K and $Succ(K)$. If n_X coincides with n_K, then X is aggregated to K by merging $K(X)$ and K.
- *Split Insertion*
  For the split insertion, we consider inserting the service X *between events*. The notion of key point K makes appear two specific events: (1) the event related to the arrival of the vehicle v at the node n_K, and (2) the event related to the departure of v from n_K. According to this, for a split insertion, we insert the event X between the arrival event of v at K and the departure event at K. The semantics go as follows: v arrives at the node n_K, drops off its outbound passengers whose destination is n_K, moves

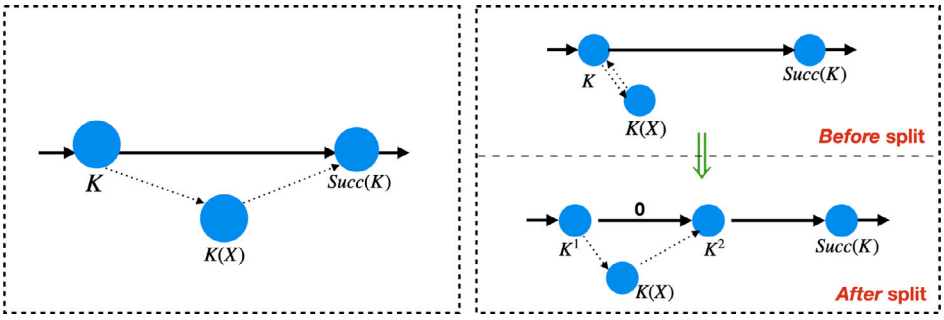


Fig. 2. Regular Insertion (left) and Split Insertion (right).

to n_X, performs the service X, and finally comes back to n_K to pickup the inbound passengers whose origin is n_K. Notice that the split insertion makes sense only when there are both inbound and outbound requests aggregated at the same time in K, because otherwise v would have no reason to return to n_K. Algorithmically, for a split insertion, we first split key point K into two new key points K^1 and K^2, with K^1 keeping only the arrival events of K, and K^2 keeping only the departure events of K, disentangling as such the time windows and load attributes according to Algorithm 1. Then $K(X)$ is inserted between K^1 and K^2 as shown by the lower figure on the right in Fig. 2. If $n_X = n_K$, then we consider that the split insertion does not have any operational semantics, and so we do not allow it.

Algorithm 1 Rules: split the key point K

Adjust the service actions:

$R^+_{K^1} \leftarrow \varnothing$, $q_{K^1} \leftarrow q_K - \sum_{r \in R^+_K} q^r$ ▷*We only keep the outbound requests on* K^1

$R^-_{K^2} \leftarrow \varnothing$ ▷*We only keep the inbound requests on* K^2.

Adjust the time windows:

$e^d_{K^1} \leftarrow e^a_K$ ▷*The earliest departing time no longer depends on the earliest in-vehicle time of the passengers.*

$l^a_{K^2} \leftarrow l^d_K$ ▷*The latest departing time no longer depends on the maximum ride time of the passengers.*

Above, we introduced *local insertion* of services. The detailed description on how time windows are determined on $K(P^r)$ and $K(D^r)$ can be found in Appendix A. Then, a *global insertion* of r into θ^v consists therefore in the composition of local insertion of services: We first insert P^r then D^r, while applying in each case either *regular insertion* or *split insertion*. It requires the knowledge of the following parameters: $K^p, \lambda_{K^p}, K^d, \lambda_{K^d}$, where $K^p, K^d \in \theta^v$ are respectively the insertion positions of P^r and D^r, and $\lambda_{K^p}, \lambda_{K^d} \in \{regular, split\}$ are respectively the insertion types for P^r at K^p and for D^r at K^d. Consequently, the resulting global insertion procedure *G_INSERTION* takes the request r, together with a 5-tuple parameter $(v, K^p, \lambda_{K^p}, K^d, \lambda_{K^d})$. Effectively performing it requires checking that the load of v between $K(P^r)$ and $K(D^r)$ never exceeds the capacity Q. Besides, it also requires updating the time windows of key points along θ^v and verifying none of the time windows become empty. The latter process involves a constraint propagation process. Since it is impacted by our transit network-based route modeling and induces an important part of the computational effort, we shall make some effort to discuss it.

4.1.2. Propagation of the time constraints

A change in the time windows of a key point $K \in \theta^v$ is likely to impact other key points in the route. For any key point $K \in \theta^v$, its time windows are subject to the following time constraints:

- $e^a_P \leqslant e^d_P$ and $l^a_P \leqslant l^d_P$, (C1)
- $e^d_P + t(K, Succ(K)) \leqslant e^a_{Succ(K)}$ and $l^d_P + t(K, Succ(K)) \leqslant l^a_{Succ(K)}$, (C2)
- $\forall r' \in R^+_K, l^a_{K(D^{r'})} - \min(l^d_K, l_{P^{r'}}) \leqslant T^{r'}$ (C3)
- $\forall r' \in R^-_K, e^a_K - \min(e^d_{K(P^{r'})}, l_{P^{r'}}) \leqslant T^{r'}$ (C4)
- $\forall r' \in R^+_K, e^a_K \leqslant l_{P^{r'}}$ (C5)

Constraints (C1) and (C2) ensures the correct order of the arrival and departure events at key points. Constraints (C3) and (C4) verifies that the maximum in-vehicle time constraint of requests are always satisfied. In (C3), the term $\min(l^d_K, l_{P^r})$ computes the latest in-vehicle time of r: if $l_{P^r} < l^d_K$, requests are allowed to get in the vehicle at l_{P^r} and wait until l^d_K before v departs. Similarly, in (C4), the term $\min(e^d_{K(P^r)}, l_{P^r})$ computes the earliest in-vehicle time of r: if v departs at $e^d_{K(P^r)}$ earliest from the pickup location of r, then $e^d_{K(P^r)}$ is the earliest boarding time of r; otherwise, r boards at l_{P^r} and waits until $e^d_{K(P^r)}$ before departing. Finally, Constraint (C5) verifies that the vehicle must arrive at K before the latest pickup time of all its inbound requests let them get onboard on time.

Each of the above constraint gives rise to a constraint propagation rule:

Constraint (C1) is violated: For the earliest time violation, we update e^d_P by $e^d_P \leftarrow e^a_P$; for the latest time violation, we update l^a_P by $l^a_P \leftarrow l^d_P$;

Constraint (C2) is violated: For the earliest time violation, we update $e^a_{Succ(K)}$ by $e^a_{Succ(K)} \leftarrow e^d_K + t(K, Succ(K))$; for the latest time violation, we update $l^d_{Pred(K)}$ by: $l^d_{Pred(K)} \leftarrow l^a_K - t(Pred(K), K)$;

Constraint (C3) is violated: For each inbound request $r' \in R^-_K$ in question, we increase the earliest departure time from its origin: $e^d_{K(P^{r'})} \leftarrow e^a_K - T^{r'}$;

Constraint (C4) is violated: For each outbound request $r' \in R^+_K$ in question, we decrease the latest arrival time of its destination: $l^a_{K(D^{r'})} \leftarrow l^d_K + T^{r'}$;

Constraint (C5) is violated: We stop (failure signal) since the earliest arrival time at K becomes too late compared to the latest in-vehicle times of its inbound requests.

Empty time window: If any time window becomes empty or negative during the propagation, the process is stopped (failure signal) as the current time windows and the above-listed constraints become inconsistent.

To efficiently cast these rules into a constraint propagation process, we regard each modification on the timestamp as a trigger, and use a list $\mathscr{L}$ to store these triggers. We deal with triggers in $\mathscr{L}$ one by one and continuously add new modifications to $\mathscr{L}$ until all triggers have been processed and no more updates are due, or until we receive a failure signal.

We notice that, thanks to our route modeling approach which greatly reduces the length of routes, the processing time related to Constraints (C1) and (C2) can be significantly improved compared to that of the standard approach. However, as in the standard modeling, the processing time related to constraint (C3) and constraint (C4) keeps contributing in $O(R(v))$, where $R(v)$ is the number of requests involved in the route θ^v. The following trick can be introduced to further decrease the computational effort: If $e^d_{K(P^r)} + T^r \leqslant l^a_{K(D^r)}$ is satisfied for a request r, then Constraints (C3) and (C4) are trivially satisfied and thus can be *neglected*, meaning that we no longer need to proceed with the related rules regarding r. Since T^r is usually large compared to the size of the vehicle's passage time windows, this situation will most often hold.

4.2. Insertion order strategy

The processing order of the requests may have a strong impact on the behavior of the DAR system. In this paper, we try several methods:

- *Random:* The request to be inserted at any time during the process is randomly selected.
- *TWP (Time Window Pickup):* Requests are ordered according to the increasing earliest in-vehicle time e_{Pr}.
- *STCluster (Spatial-Temporal Cluster):* The network $\mathcal{G}$ is divided into disjoint areas, and the temporal space is divided into time slots. Then, requests with the same departure time slot and the same pickup and drop-off areas are grouped into the same cluster. Requests in the same cluster are further sorted according to *TWP*. During the process, we explore these clusters while following the increasing order of time slots, and consecutively insert requests in the same cluster. The intuitive purpose of this approach is that requests with similar departing times and close pickup and drop-off locations should be served by the same SAVs.

The impact of different strategies is studied in Section 8.

5. Search_Insert: Algorithmic scheme

The *Search_Insert* procedure is the core issue of our **LS_DARP**. It takes the request r and the current route collection Θ as input and searches for well-fitted parameters $(v^*, K^{p*}, \lambda^*_{K^p}, K^{d*}, \lambda^*_{K^d})$ for the effective insertion $G\_INSERTION(r, v^*, K^{p*}, \lambda^*_{K^p}, K^{d*}, \lambda^*_{K^d})$ into θ^v.

The standard approach involves enumerating all the vehicles v, then for every v, enumerating the 4-tuple $\{K^p, \lambda_{K^p}, K^d, \lambda_{K^d}\}$, and finally selecting $(v^*, K^{p*}, \lambda^*_{K^p}, K^{d*}, \lambda^*_{K^d})$ that minimizes the increase in the operational cost as the best-fitted insertion parameters of r. However, in a large-scale context, this approach requires too much computational time and is thus especially unadaptable in a dynamic situation. For that, we integrate a *filtering system* into *Search_Insert* that quickly identifies well-fitted insertion parameters to avoid exhaustive enumerations.

In this section, we briefly give an overview of the filtering system (the detailed description of which is presented in Section 6) before explaining the role it plays inside the general architecture of *Search_Insert*.

The filtering system essentially relies on:

- Two specific data structures: the *vehicle filtering matrix* and the *insertion position filtering matrix*, which link the routes, the requests, and a subdivision of the time space and the transit networks into periods and zones;

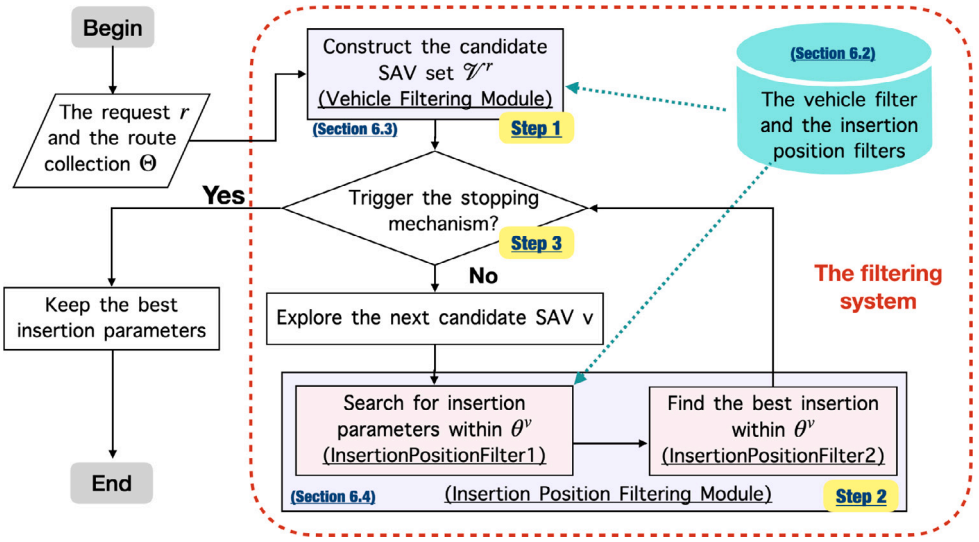


Fig. 3. The algorithmic scheme for the procedure *Search_Insert*.

- Two specific modules: *vehicle filtering module* and *insertion position filtering module*, which work in cascade in order to identify the best-fit parameters for the insertion of the target request r;
- A *stopping mechanism* that controls the number of calls for the time-consuming constraint propagation process.

Given r and Θ, the algorithmic scheme for *Search_Insert* (see Fig. 3) is designed as follows:

Step 1 - Selecting and Ranking the Vehicle Candidates: We first invoke a *vehicle filtering module*. With the help of a *vehicle filtering index matrix*, this filtering unit extracts from $\mathcal{V}$ a group $\mathcal{V}^r$ of candidate vehicles that are likely to be able to efficiently serve r. Then we assign each candidate vehicle $v \in \mathcal{V}^r$ with an *insertion score* that provides a heuristic estimation of the feasibility of using v for request r.

Step 2 - Selecting and Trying the Insertion Position Parameters: Once set $\mathcal{V}^r$ is constructed, we explore the candidate vehicles v according to the decreasing order of their insertion scores and invoke an *insertion position filtering module*, which relies on a collection of *insertion position filtering matrices*. This filtering unit further contains two procedures: *InsertionPositionFilter1* and *InsertionPositionFilter2*. *InsertionPositionFilter1*(r, v) identifies, for a candidate $v \in \mathcal{V}^r$, the most valuable pairs (K^p, λ_{K^p}) and (K^d, λ_{K^d}) allowing the respective insertions of P^r and D^r without violating their time windows. The output of this procedure is two lists of candidate position parameters: $\mathcal{L}^v_{Pr} = \{\dots, (K^p, \lambda_{K^p}), \dots\}$ and $\mathcal{L}^v_{Dr} = \{\dots, (K^d, \lambda_{K^d}), \dots\}$.

If both $\mathcal{L}^v_{Pr}$ and $\mathcal{L}^v_{Dr}$ are non-empty, they are passed to the second procedure *InsertionPositionFilter2*$(r, v, \mathcal{L}^v_{Pr}, \mathcal{L}^v_{Dr})$. It performs the most time-consuming task since it simulates, for $v \in \mathcal{V}^r$, the insertion of r with respect to all combinations of parameters from $\mathcal{L}^v_{Pr}$ and $\mathcal{L}^v_{Dr}$. Namely, it first performs some routine tests on time windows and loads before, in the event of success, performing the constraint propagation process described earlier in Section 4.1.2. At the end of the process, among all feasible 5-tuple $(v, K^p, \lambda_{K^p}, K^d, \lambda_{K^d})$, we keep the one with the minimum insertion cost.

Step 3 - Stop or Keep Going?: The last key component of the *Search_Insert* is its stopping mechanism: we must absolutely avoid exploring too many vehicles and too many insertion parameters $K^p, K^d, \lambda_{K^p}, \lambda_{K^d}$. This stopping mechanism involves two components:

1. **A *Computational Effort* Counter**: We control the computational effort due to the current call to the *Search_Insert*$(r, \mathcal{V})$ procedure by introducing a control threshold parameter ϕ and an *effort counter* φ. The variable ϕ defines the maximum number of calls to the constraint propagation procedure, and φ keeps track of the current number of calls to that procedure.
2. **A *Regeneration* Mechanism**: As soon as φ reaches the threshold ϕ without yielding any feasible insertion, we may reset it to 0 and start again while restricting ourselves to the vehicles of $\mathcal{V}^r$ that have not been tried yet. The way the regeneration mechanism works is commanded by an iteration number *iter*: We increment *iter* every time we reset φ, and the whole process stops (in case we register no success) when *iter* exceeds some threshold $Iter\_max$. In practice, $Iter\_max = 1$, meaning that the regeneration action can be triggered at most once for each request.

Algorithm 2 summarizes the above described process.

Algorithm 2 *Search_Insert*(r, Θ)

$found \leftarrow false$; $(v^*, K^{p*}, \lambda^*_{K^p}, K^{d*}, \lambda^*_{K^d}) \leftarrow NaN$; $cost^* \leftarrow \infty$
$\mathcal{V}^r \leftarrow$ VehicleFilter$(r, \mathcal{V})$
set the values of ϕ and $Iter\_Max$ for the current insertion
$\varphi \leftarrow 0$; $iter \leftarrow 0$
while there is still $v \in \mathcal{V}^r$ to explore and ($\varphi \leqslant \phi$ or $iter \leqslant Iter\_Max$) **do**
  $\mathscr{L}^v_{P^r}, \mathscr{L}^v_{D^r} \leftarrow$ InsertionPositionFilter1(r, v)
  if $\mathcal{L}^v_{P^r} \neq \varnothing$ ς $\mathcal{L}^v_{D^r} \neq \varnothing$ **then**
    $found, cost^v, K^p, \lambda_{K^p}, K^d, \lambda_{K^d}, \varphi^v \leftarrow$ InsertionPositionFilter2$(r, v, \mathscr{L}^v_{P^r}, \mathscr{L}^v_{D^r})$
    $\varphi \leftarrow \varphi + \varphi^v$
    if $\varphi > \phi$ **then**
      $\varphi \leftarrow 0$
      $iter \leftarrow iter + 1$
    if $found$ ς $cost^v < cost^*$ **then**
      $(v^*, K^{p*}, \lambda^*_{K^p}, K^{d*}, \lambda^*_{K^d}) \leftarrow (v, K^p, \lambda_{K^p}, K^d, \lambda_{K^d})$
      $cost^* \leftarrow cost^v$
      $iter \leftarrow Iter\_Max + 1$
return $(v^*, K^{p*}, \lambda^*_{K^p}, K^{d*}, \lambda^*_{K^d})$

6. Filtering system

An SAV v is eligible to serve r only if a visit to the pickup location p^r during $[e_{P^r}, l_{P^r}]$ and a visit to the destination d^r without violating T^r can be inserted into its schedule. However, as described earlier, finding SAVs that satisfy the above conditions is too time-consuming in the large-scale context. Therefore, the filtering system is conceived based on the idea to quickly obtain a set of candidates which "roughly" satisfy the above conditions instead: "We search for vehicles that may easily reach the spatial neighborhood (the notion of *zone*) of the target locations during the temporal neighborhood (the notion of *period*) of the related time windows".

In this section, we present a detailed description on different components of the filtering system that efficiently drives us towards well-fitted insertion parameter 5-tuples $(v, K^p, \lambda_{K^p}, K^d, \lambda_{K^d})$. We begin by defining in Section 6.1 a spatial–temporal preprocessing and in Section 6.2 two types of filtering devices: the vehicle filtering matrix and the insertion position filtering matrices. Then two filtering modules, the vehicle filtering module and the insertion position filtering module, are explained in order in Sections 6.3 and 6.4. With the help of the two filtering matrices that rely on the spatial–temporal preprocessing, the three filtering modules quickly select valuable SAVs and insertion positions, and identify the best insertion among all the insertion parameters.

6.1. Spatial-temporal preprocessing: Zones and periods

Zones. We use a grid to divide the transit network into a set of *zones*, $\mathcal{Z} = \{0, 1, \dots, z, \dots, Z-1\}$. For each node $n \in \mathcal{N}$, we denote the zone where n belongs by z_n. For the sake of simplicity, for a key point K, the zone where n_K belongs is marked as z_K instead of z_{n_K}. Similarly, for pickup and drop-off services of a request r, P^r and D^r, the corresponding zones are denoted, respectively, by z_{P^r} and z_{D^r}. Notice that the *zones* do not necessarily need to be the same as the spatial areas involved in the *STCluster* insertion order strategy.

We need, while performing the filtering processes, to link the pickup location p^r and the drop-off location d^r of any request r to the zones defined this way. For that, we define a *Node-Zone Travel Time Estimator* whose purpose is to estimate how easy it may be for a vehicle v to leave one of its key points K, perform some new service located in a zone z before coming back to its successive key point. The estimator takes a node n of $\mathcal{G}$ and a zone z, and returns an estimate $\hat{t}(n, z)$ ($\hat{t}(z, n)$) of the travel time from n to z (from z to n). Notice that we do not specifically want the resulting estimate to be optimistic (to define a lower bound of the time that the target vehicle v may spend inside z), but rather provide us with a rough evaluation of the plausibility of such a detour. Specifically, to measure these estimates, we define four reference points within the zone z, chosen as the midpoint of each edge of the zone. Let $\{y^1_z, y^2_z, y^3_z, y^4_z\}$ be the set of these four references (see Fig. 4). Then we set:

$$\hat{t}(n, z) = \hat{t}(z, n) = \begin{cases} \min_{1 \leqslant i \leqslant 4} t^{euc}(n, y^i_z) & \text{if } z_n \neq z \\ 0 & \text{otherwise,} \end{cases}$$

where $t^{euc}(n, y)$ is the travel time in terms of Euclidean distance. From empirical experiments, such a definition coordinates well with our filtering system. For a key point K and a zone z, we denote by $\hat{t}(K, z)$ and $\hat{t}(z, K)$ the respective estimated node-zone and zone-node travel times.

Periods. We equally divide the time horizon T into a set of periods $\mathcal{H} = \{0, 1, \dots, h, \dots, H-1\}$. Denoting the beginning timestamp of the period h by t_h, and the length of each period by $I_{\mathcal{H}}$, we have

$$t_h = h \cdot I_{\mathcal{H}},$$

and

$$I_{\mathcal{H}} \cdot H = T.$$

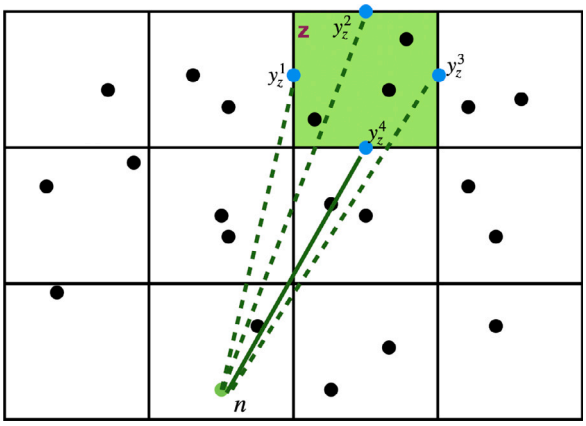


Fig. 4. The node-zone travel time estimator. In the above example, $\hat{t}(n,z) = t^{euc}(n, y_z^4)$.

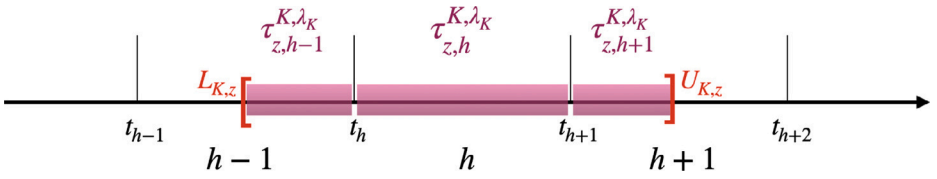


Fig. 5. Elastic durations for K at the zone z.

In addition, given a date t, we categorize it into the time period $h(t)$, with

$$h(t) = \lfloor t / I_{\mathcal{H}} \rfloor.$$

Furthermore, for any request r, we denote by $\mathcal{H}_{P^r}$ the smallest set of consecutive periods (*period interval*) covering the pickup time window, i.e.,

$$\mathcal{H}_{P^r} = \{h \in \mathcal{H} : h(e_{P^r}) \leqslant h \leqslant h(l_{P^r})\}. \tag{1}$$

The period interval of D^r is defined in the same way.

6.2. The filtering matrices

We now introduce two types of 2-index matrices, $M[\mathcal{Z} \times \mathcal{H}]$ and $M^v[\mathcal{Z} \times \mathcal{H}]$. The former is the vehicle filtering matrix, and the latter is the insertion position filtering matrix regarding the vehicle $v \in \mathcal{V}$. These filtering matrices together provide a projection of the activity of the vehicles onto the periods and the zones.

The insertion position filtering matrix. With any active vehicle v, we associate a matrix M^v of size $|\mathcal{Z}| \times |\mathcal{H}|$ called its *Insertion Position Filtering* matrix. Given a zone-period pair $(z,h) \in \mathcal{Z} \times \mathcal{H}$, $M^v[z,h]$ contains a list of triplets $(K, \lambda_K, \tau^{K,\lambda_K}_{z,h})$, where $K \in \theta^v$ represents a key point in θ^v after which it is estimated that v might move through zone z during period h, $\lambda_K = \{regular, split\}$ indicates the insertion type, and $\tau^{K,\lambda_K}_{z,h}$ is an estimation, called *elastic duration*, of the time available for an insertion of type λ_K involving z and h. For any vehicle v, any zone z and any period h, $(K, \lambda_K, \tau^{K,\lambda_K}_{z,h})$ are computed as follows:

- For $\lambda_K = regular$, we estimate the maximum time interval $[L_{K,z}, U_{K,z}]$ during which v may pass through the zone z from K and to its successor $Succ(K)$. Specifically, v can reach z no earlier than $L_{K,z} = e^d_K + \hat{t}(K,z)$ and should leave z no later than $U_{K,z} = l^a_{Succ(K)} - \hat{t}(z, Succ(K))$.
- Then, for each period h, $\tau^{K,regular}_{z,h}$ is given by (see Fig. 5):

$$\tau^{K,regular}_{z,h} = \min(U_{K,z}, t_{h+1}) - \max(L_{K,z}, t_h). \tag{2}$$

- We do the same for $\lambda_K = split$, while dealing with the time that v may spend in z between its arrival into n_K and the time when it leaves n_K. Then we get a $\tau^{K,split}_{z,h}$.
- $M^v[z,h]$ stores all 3-tuple $(K, \lambda_K, \tau^{K,\lambda_K}_{z,h})$ such that $\tau^{K,\lambda_K}_{z,h} > 0$.

The vehicle filtering matrix. The Vehicle Filtering Matrix M may be viewed as both an aggregation and a reciprocal image of the $M^v[z,h]$ matrices. At any time during the *ProcessDARP* process, it derives from the matrices M^v the vehicles that are able to pass in a zone in a given period, together with some grade(s) $\mathcal{T}_{v,z,h}$, that we call the *time flexibility(ies)* of the vehicles. Those *time flexibilities* estimate the fitness of a vehicle v with any tentative insertion involving a service P^r or D^r located in the area z and with a time window at h. So, for any $(z,h) \in \mathcal{Z} \times \mathcal{H}$, $M[z,h]$ means a list of pairs $(v, \mathcal{T}_{v,z,h})$ such that:

- $M^v[z,h] \neq \emptyset$;
- Numerical parameter $\mathcal{T}_{v,z,h}$ is defined by

$$\mathcal{T}_{v,z,h} = \bigoplus_{(K,\lambda_K,\tau_{z,h}^{K,\lambda_K}) \in M^v[z,h]} \tau_{z,h}^{K,\lambda_K}, \tag{3}$$

  where $\oplus$ is an associative operator that applies to the elastic durations of the key points of θ^v in $M^v[z,h]$.

We consider two operators $\oplus$: *sum* and *max*, which means that $\mathcal{T}_{v,z,h}$ may refer to two values. In any case $\mathcal{T}_{v,z,h}$ is called the *time flexibility* of the vehicle, in zone z in period h. We understand that a high value $\mathcal{T}_{v,z,h}$ means a higher likelihood of feasibility of inserting a new service at z during h into the route of v, and so that we are going to use this notion of *time flexibility* as an empirical tool in order to rank the vehicles with respect to a potential insertion involving the zone z and the period h. The fact that we simultaneously rely on two operators $\oplus$ corresponds to the fact that those operators tend to favor different classes of activity patterns: the operator *sum* favors the vehicles moving several times around or inside the area z during the period h; the operator *max* tends to give a chance to vehicles that are currently idle during period h and located far from z.

Updating the filtering matrices. The filtering matrices M and M^v are updated every time a request is inserted into a vehicle v. While performing this process, we distinguish two types of key points in θ^v: those that are newly inserted and those that previously existed. For any newly created key point K, we compute all the zones that can be reached from this point and the corresponding arrival periods. Then, we add K to the related cells in M^v, and update the time flexibility of v in the same cells of M. As for the previously existing key points, their time windows may be tightened by the insertion of a new request. As a consequence, the zones that can be reached from these affected points may decrease. Then we should remove these points from the corresponding cells in M^v. In addition, due to the insertion of the new service, the vehicle may become fully occupied on some points K, such that $q_K = Q$. Then we shall remove these points from all cells in M^v. Finally, if no feasible key points are left regarding a zone and a period, we should delete v from the corresponding cell of the vehicle filtering matrix M.

6.3. The vehicle filtering module

An SAV v is considered a *candidate vehicle* for the target request r if it is estimated to be capable of serving r. In this subsection, we present how to select valuable candidate vehicles using the vehicle filtering matrix M.

To assign r to an SAV v means to insert the pickup service P^r and the drop-off service D^r into θ^v. For $X \in \{P^r, D^r\}$, we compute a *local score* $s_{v,X}^{local}$, that intuitively estimates the *plausibility* of a feasible insertion of the service X into θ^v. This local score $s_{v,X}^{local}$ is defined as the weighted sum, over the period interval $\mathcal{H}_X$ of X, of the time flexibilities of v to reach the zone z_X over the $\mathcal{H}_X$:

$$s_{v,X}^{local} = \sum_{h \in \mathcal{H}_X} \beta_{h,X} \mathcal{T}_{v,z_X,h}, \tag{4}$$

where the weight $\beta_{h,X}$, representing the contributions of every h to the time window of X, is defined by

$$\beta_{h,X} = \frac{\min(l_X, t_{h+1}) - \max(e_X, t_h)}{l_X - e_X}. \tag{5}$$

This way of defining the score $s_{v,X}^{local}$ follows the idea that a vehicle with a large time flexibility is more likely to enable the insertion. Let us recall that there can be two values of time flexibility $\mathcal{T}$ according to the operator $\oplus \in \{sum, max\}$, where *sum* favors the vehicles that are better occupied with more key points and *max* favors idle vehicles. The local score $s_{v,X}^{local}$ may thus have two values. Specifically, we use both operators to compute s_{v,P^r}^{local}; while we always use the same operator *max* for the score s_{v,D^r}^{local}, taking into account that the time window of D^r is most often significantly larger than the time window of P^r, making the insertion of the pickup service more difficult than the insertion of the drop-off service.

Then an SAV v is considered a candidate vehicle only if both s_{v,P^r}^{local} and s_{v,D^r}^{local} are positive. Intuitively, a well-fitted candidate vehicle should exhibit a high local score on both its pickup and drop-off services. Consequently, we define the *global insertion score* $s_{v,r}$ of v with respect to r as:

$$s_{v,r} = \min(s_{v,P^r}^{local}, s_{v,D^r}^{local}). \tag{6}$$

Notice that s_{v,P^r}^{local} and s_{v,D^r}^{local} may not be computed with the same operator $\oplus$. That is why we normalize these estimators so that they take comparable values on both sides of above min formula.

The resulting candidate vehicle set $\mathcal{V}^r$ returned by the vehicle filtering module is a set of vehicles v such that $s_{v,r} \neq 0$, ranked according to decreasing order of $s_{v,r}$. More precisely, $\mathcal{V}^r$ is the union of two sets $\mathcal{V}_{\oplus}^r$, $\oplus \in \{sum, max\}$, where $\oplus$ corresponds to the operator that we use to compute the local score s_{v,P^r}^{local}.

6.4. The insertion position filtering module

A vehicle $v \in \mathcal{V}^r$ being given, the first part of the insertion position filtering module, i.e., the procedure *InsertionPositionFilter1*(r, v), is invoked to independently search for feasible insertion positions for $X = P^r$ and $X = D^r$. At the end of the filtering process, two lists should be obtained: $\mathcal{L}^v_{P^r}$ and $\mathcal{L}^v_{D^r}$. Each element in these lists is a pair that specifies the insertion position $K \in \theta^v$ and the insertion type $\lambda_K \in \{regular, split\}$. Note that if both types of insertion are feasible at $K \in \theta^v$, then both $(K, regular)$ and $(K, split)$ will be in list $\mathcal{L}^v_X$.

We get those lists $\mathcal{L}^v_{P^r}$ and $\mathcal{L}^v_{D^r}$ by scanning the lists $M^v[z_X, h]$, $h \in \mathcal{H}_X$ and checking, for each pair $(K, \lambda_K) \in M^v[z_X, h]$, that X can be inserted at K into θ^v according to the insertion type λ_K. Specifically, if (K, λ_K) satisfies the following constraints, it is added to the list of insertion positions $\mathscr{L}^v_X$.

For $\lambda_K = regular$, the following inequalities must be satisfied:

$$q_K + q^r \leqslant Q \qquad \text{if } X = P^r \tag{7}$$

$$e^d_K + t(K, X) \leqslant l_X \tag{8}$$

$$e_X + t(X, Succ(K)) \leqslant l^a_{Succ(K)} \tag{9}$$

$$e^d_K + t(K, X) \leqslant l^a_{Succ(K)} - t(X, Succ(K)) \tag{10}$$

where $t(K, X)$ ($t(X, K)$) denotes the travel time from n_K to the location of X (from the location of X to K). Constraint (7) means that the number of passengers must not exceed the vehicle capacity. Constraint (8) means that the vehicle must arrive no later than the upper bound of the time window defined for the service X. Constraint (9) checks that the insertion of X does not violate the upper bound on the arrival time at the successive key point $Succ(K)$. Finally, Constraint (10) ensures the vehicle can arrive at X before having to leave.

For $\lambda_K = split$, the following inequalities must hold:

$$q_K - \sum_{r' \in R^+_K} q^{r'} + q^r \leqslant Q \qquad \text{if } X = P^r \tag{11}$$

$$e^a_K + t(K, X) \leqslant l_X \tag{12}$$

$$e_X + t(X, K) \leqslant l^a_K \tag{13}$$

$$e^a_K + t(K, X) \leqslant l^a_K - t(X, K) \tag{14}$$

Conditions (11)–(14) correspond to the same constraints as conditions (7)–(10), but within the context of split insertion, where the vehicle should be able to leave from K to provide the new service X within the time window of X, and return to K.

If both $(K, regular)$ and $(K, split)$ are in $M^v[z_X, h]$, we can speed this testing process by noticing that conditions (11), (12) and (9) are necessary conditions for (7), (8) and (13) (see Appendix C for proofs). Thus, we merge the verification processes related to the two insertion types, and first test (11), (12) and (9) as an outer control before trying the other inequalities.

Finally, we invoke the procedure *InsertionPositionFilter2*$(r, v, \mathscr{L}^v_{P^r}, \mathscr{L}^v_{D^r})$, once the lists $\mathcal{L}^v_{P^r}$ and $\mathcal{L}^v_{D^r}$ have been computed. We test the feasibility of all resulting 5-tuple $(v, K^p, K^d, \lambda_{K^p}, \lambda_{K^d})$ and keep, among those feasible 5-tuples, the one that minimizes the detour cost. Once again, we perform a *cascade* process to ensure efficiency.

Joint Insertion Test: For any insertion pair $((K^p, \lambda_{K^p}), (K^d, \lambda_{K^d})) \in \mathscr{L}^v_{P^r} \times \mathscr{L}^v_{D^r}$, with K preceding K^d, we first verify whether P^r and D^r can be inserted simultaneously by analyzing the necessary (but not sufficient) maximum ride time constraint of r:

$$t^* + t(K^d, D^r) \leqslant \min(l_{P^r}, t^\dagger) + T^r, \tag{15}$$

The left-hand side of the equation computes the lower bound of the arrival time at D^r. If λ_{K^d} is a regular insertion, $t^* = e^d_Q$; otherwise, $t^* = e^a_Q$. The right-hand side of the equation computes the upper bound timestamp for r to get to its destination considering its maximum ride time T^r. If λ_{K^p} is a regular insertion, $t^\dagger = t^a_{Succ(K)} - t(P^r, Succ(K))$; otherwise, $t^\dagger = l^d_K - t(P^r, K)$. This term calculates the latest departure time from P^r. The timestamp $\min(l_{P^r}, t^\dagger)$ computes the upper bound of r's in-vehicle time. If l_{P^r} is earlier than the vehicle's latest departure time from P^r, r is allowed to get onboard at l_{P^r} and wait until v is ready to depart, as long as the maximum in-vehicle time T^r is respected.

Load Test: We check, for any intermediate key point K^* between K and K^d, whether the following inequality holds:

$$q_{K^*} + q^r \leqslant Q. \tag{16}$$

Constraint Propagation Test: The two previous tests correspond to necessary but not sufficient conditions for the insertion 5-tuple $(v, K^p, \lambda_{K^p}, K^d, \lambda_{K^d})$ to be valid. In order to ensure the feasibility of the procedure *G_INSERTION*$(v, K^p, \lambda_{K^p}, K^d, \lambda_{K^d})$, we must apply the constraint propagation procedure. As explained in Section 5, this last component of the filtering system controls the times allowed to the *Search_Insert* procedure for the computation of some feasible insertion 5-tuple $(v, K^p, \lambda_{K^p}, K^d, \lambda_{K^d})$. Every time we launch the constraint propagation procedure, we increase by 1 the *effort counter* φ that commands the main loop of *Search_Insert*, together with the *regeneration* counter *iter*.

7. Extension to a dynamic context

Dealing with a dynamic context is in practice a difficult issue. Our *ProcessDARP* algorithm should work in real time, in a way that is synchronized with both the emission of the requests and the continuous updating of the knowledge related to the current state of the fleet $\mathcal{V}$. Besides, its design should take into account the protocols that rule the communication between the vehicles, the central platform and the users, as well as the uncertainty related to the behavior of the different players (delays, defaults, etc.). Finally, deciding when *ProcessDARP* has to be launched also becomes part of the problem. We only address here a simplified dynamic setting of our **LS_DARP** problem. We care neither about the communication protocols nor the real-time issues, and we characterize our dynamic setting by the following features:

1. Every request r is provided with an *emission time* $t^r_{emit} \leq e_{P^r}$: we suppose that t^r_{emit} and $e_{P^r} - t^r_{emit}$ follow independent random laws. Clearly, requests must be handled according to increasing t^r_{emit} values.
2. We define a set of decision epochs $D = \{D_0, D_1, \dots, D_i, \dots, D_{|D|-1}\}$. Each decision epoch lasts I_d (for instance $I_d = 10$ min) time units. The starting time of the epoch D_i is $t_{D_i} = i \times I_d$. During the time interval $[t_{D_i}, t_{D_i} + \mu]$, $\mu < I_d$, *ProcessDARP* processes the requests r emitted between $t_{D_{i-1}}$ and t_{D_i}. Notice that this means a simplification with respect to real contexts, where the characteristics of the epochs may be adapted to the density of the request flow.
3. At time $t = t_{D_i} + \mu$, the system communicates with passengers whose requests have been allocated to an SAV, providing more accurate details regarding the vehicle's arrival times at their pickup locations. Following this update, SAVs start implementing the new routes until reaching $t = t_{D_{i+1}} + \mu$.
4. A realistic dynamic setting should involve a vehicle fleet with a fixed size and possibly force the system to reject some requests, making the number of rejected requests part of the objective function. However, in order to link together our static and dynamic settings, we artificially act as if there were no limit on the number of available vehicles and keep on with the same objective function as in the static case.

According to this, simulating the global insertion process can be done through the application of the following *Dynamic_ProcessDARP* algorithm 3:

Algorithm 3 Dynamic_ProcessDARP($\mathcal{R}$)

$\mathcal{V} \leftarrow \emptyset, \Theta \leftarrow \emptyset$
Sort $\mathcal{R}$ according to increasing *emission time* values t^r_{emit}
For any $D_i = D_0, D_1, \dots, D_{|D|-1}$, set $\mathcal{R}_{D_i} = \{r \in \mathcal{R} \quad s.t \quad t_{D_{i-1}} \leq t^r_{emit} < t_{D_i}\}$ ▷*for* D_0, $\mathcal{R}_{D_0}$ *is formed by static requests if there are any.*
for $i = 0 \dots, |D| - 1$ **do**
  while there are still unexplored requests in $\mathcal{R}_{D_i}$ **do**
    Sort $\mathcal{R}_{D_i}$ according to *STCluster*.
    $(v^*, K^{p*}, \lambda^*_{K^p}, K^{d*}, \lambda^*_{K^d}) \leftarrow$ SEARCH_INSERT($r, \mathcal{V}$) ▷*Search for well-fitted insertion parameters*
    if $Fail(Search\_Insert)$ **then**
      activate a new vehicle v to serve r
      add v into the currently activated SAV set $\mathcal{V}$
return Θ

8. Numerical experiments

We present several results of numerical experiments. In Section 8.1, we introduce the input data and basic problem settings. In Section 8.2, we analyze the characteristics and the performances of the underlying SAV DAR system. In Section 8.3, we show the impacts of different algorithmic choices on the basic resolution of the problem. Then, in Section 8.4, we evaluate the effectiveness of the filtering system and the insertion strategy. And in Section 8.5, we explore the potential of our algorithm in a dynamic context.

We programmed the algorithms in C++ language and solved the problem on a 512 GB RAM machine with an AMD EPYC 7452 32-Core Processor.

8.1. Input data

We take the network of the city of Clermont-Ferrand, France, and its suburbs. The input data of the network is composed of the information of the nodes and arcs, generated by a graph generation tool using the original data from OpenStreetMap. The selected service area has an approximate surface of 23 km × 21 km, with 13,839 nodes and 31,357 arcs. Every node is identified with a type. In this work, we only keep two of them: **W** (working place) and **R** (residential place). All nodes with other types are unified to **U** (unclassified). Among all the nodes, we proportionally keep 1460 to serve as pickup and drop-off locations (Fig. 6). The percentage of effective pickup and drop-off locations in each type of selected network is shown in Table 2.

The system we study in this paper is still prospective, so it is impossible to comprehensively understand the passengers' travel needs for such a system at this state, and it would be inappropriate to use the data set extracted from real-life transportation systems. We developed a simple tool to generate requests that roughly comply with the intended use cases of this system: providing services to all kinds of travel requests. We explain how to generate requests for weekdays, when daily commute demands typically dominate travel demands. For weekend requests, it is only a matter of adjusting certain distributions introduced in the following.

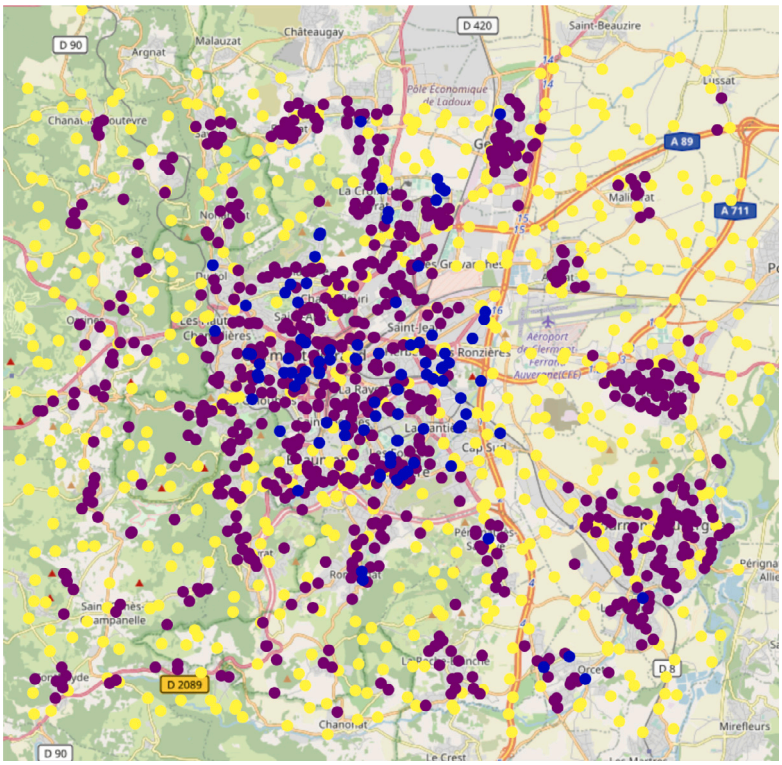


Fig. 6. The road network of Clermont-Ferrand: the city center and its rural areas. Nodes in the figure represent the effective pickup and drop-off locations: blue nodes are of type **W**, purple nodes are of type **R**, and yellow nodes are of type **U**.

Table 2
The statistics of node classification.

Node type	U	R	W
Count	574	788	98
Percentage	39.3%	54.0%	6.7%

Table 3
The description of five time slots.

Time slot	Duration	Request proportion
Morning Slack (MS)	00:00~06:00	1%
Morning Peak (MP)	06:00~10:00	35%
Normal Hours (NH)	10:00~15:00	20%
Evening Peak (EP)	15:00~19:00	35%
Evening Slack (ES)	19:00~24:00	9%

The service lasts $T = 24$ h, from 0:00 to 24:00. As described in Table 3, we divide the entire service period into five time slots. Each time slots contains requests with diverse travel intentions. During MP (Morning Peak), a predominant 85% of requests originate from non-working locations (**R** or **U**), with an equal percentage heading towards non-residential areas (**W** or **U**). Specifically, about half of MP requests represent *"typical"* movements from residential (**R**) to working places (**W**). During EP (Evening Peak), a symmetrical pattern is considered, with half of requests being "typical", moving from a node **W** to a node **R**. Meanwhile, requests during MS (Morning Slack), NH (Normal Hours), and ES (Evening Slack) time slots exhibit uniform distribution across various origins and destinations.

To generate a particular request r for a given time slot, its pickup location p^r, drop-off location d^r, and the earliest pickup time e_{Pr} are randomly selected according to the characteristics of the time slot. In addition, according to our no-rejection assumption, e_{Pr} also satisfies

$$e_{Pr} > t(n_0, p^r)$$

and

$$e_{Pr} < T - t(p^r, d^r) - t(d^r, n_0),$$

Table 4
Basic problem settings.

Parameter	Value
Number of pickup/drop-off locations	1460
Service beginning time t_0	00:00
Time horizon T	24 h
Vehicle capacity Q	10
Pickup time window δ	15 min
Maximum ride time amplifier α	2

so that we ensure that it is always possible to activate an unused SAV parked at the depot to service r. The length of the pickup time window for all requests is fixed as a constant number δ ($\delta = 15$ min for the basic scenario), so the latest in-vehicle time l_{Pr} is

$$l_{Pr} = e_{Pr} + \delta.$$

The amplifier α to determine the maximum ride time of all requests is set to 2 for the basic test scenario. Finally, we use a Poisson distribution $Pois(1)$ to randomly set the load q^r, which is in addition bounded by 5.

We note that requests leaving for work from home during MP should be almost "symmetric" to requests returning home from work during EP. Therefore, every time we generate a typical MP request r that originates from a node p^r tagged with **R** and terminates at a node d^r tagged with **W**, a symmetric typical EP request r' originating from d^r and destined at p^r is immediately generated with a possibility of 0.8, and vice versa.

Based on the population and daily travel needs of the underlined area in Clermont-Ferrand, we have generated instances at various scales ranging from 10,000 to 300,000 requests[4] to simulate from moderate to very high utilization of the SAV DAR system. Table 3 shows the proportion of requests generated regarding different time slots.

Without further clarifications, we adopt the *STCluster05* strategy, an instance of *STCluster*, as a preliminary step to define the processing order of requests. Specifically, *STCluster05* divides the network into 20 disjoint areas and divides the temporal space into time slots of 5 min.

Table 4 summarizes the basic settings of the problem. We tested on five different instances for each problem size and calculated the final results by averaging all the results obtained.

8.2. Observations on the characteristics of the SAV DAR system

We study the characteristics and potential of the SAV DAR system with ride-sharing. For the sake of simplicity, in this section, we term our system *share*. In comparison, we consider two other transportation systems: *no-share* and *individual*. In the former system, ride-sharing is prohibited and SAVs function as taxis; in the latter, individual cars are responsible for all vehicle movements.

With different instances of passenger requests ranging in size from 10,000 to 300,000, we evaluate the performance of the *share* system using the basic approach in which the filters are not activated. We call it f_0. In this approach, the routes of all the active vehicles in $\mathcal{V}$ are systematically explored while inserting a request r. The results of *no-share* are also obtained by f_0 while disabling ride-sharing. The number of vehicles to be used in *individual* is generously estimated based on the number of one-way requests and the number of pairs of symmetric requests. The same private vehicle can service a pair of symmetric requests since they are meant to be addressed by the same user.

Table 5 displays the basic experiment results. The fleet size decreases drastically both in *share* and *no-share* compared to *individual*. As the problem size increases, the reduction in the fleet size between *individual* and *share* (or *no-share*) amplifies, demonstrating an enhanced effectiveness of our system with larger scales. In addition, the number of SAVs needed in *share* is almost the third regarding in *no-share*. In terms of the vehicle's total drive time, *share* outperforms both two other systems, thanks to the mutualization of the trajectories. Among the three systems, *no-share* exhibits the longest total drive time, excessive time related to the empty trips made to pickup passengers compared to *individual*. We also see that, on average, each passenger spends approximately 9.9 min if no ride-sharing is allowed. In *share*, passengers' average in-vehicle time increases by around 5 min due to the detours made for ride-sharing. The above results demonstrate the effectiveness of SAV DAR systems in minimizing en-route vehicles. Notably, the considerable reduction in the required number of vehicles signifies a demand for fewer resources in vehicle production. This reduction also liberates urban space by minimizing parking needs and alleviating traffic congestion. Meanwhile, the huge reduction in vehicle drive time proves that *share* has great potential for leading to a more optimized, efficient, and environmentally friendly transportation network. In addition, the *no-share* configuration maintains passenger comfort at the sacrifice of increased operational costs tied to vehicle drive time, notably without human-related expenses, as autonomous vehicles are self-driving. Conversely, by enabling ride-sharing, *share* substantially reduces operational costs while maintaining an acceptable increase in passenger average in-vehicle time.

Furthermore, we analyze the distribution of the total drive time of vehicles across different periods while setting the number of requests at 300,000. Fig. 7 illustrates the observed curves, which demonstrates a distinct slip effect resulting from *share*. In particular,

[4] Our instances are available at https://github.com/cliu01/Dataset_LSSAVDARP_ClermontFd.

Table 5
Comparison of different systems.

$\|\mathcal{R}\|$	Fleet size			Total drive time (h)			Passenger's average in-vehicle time (min)		
	Share	No-share	Individual	Share	No-share	Individual	Share	No-share	Individual
10k	121	239	8030	1289.1	2282.1	1651.4	14.8	9.9	9.9
50k	440	1110	40,200	5121.6	10,858.0	8250.2	15.0	9.9	9.9
100k	780	2182	80,278	9324.3	21,487.6	16,515.0	15.0	9.9	9.9
200k	1385	4284	160,631	17,059.8	42,472.3	33,030.2	14.9	9.9	9.9
300k	1975	6422	240,951	24,397.6	63,570.7	49,584.9	14.9	9.9	9.9

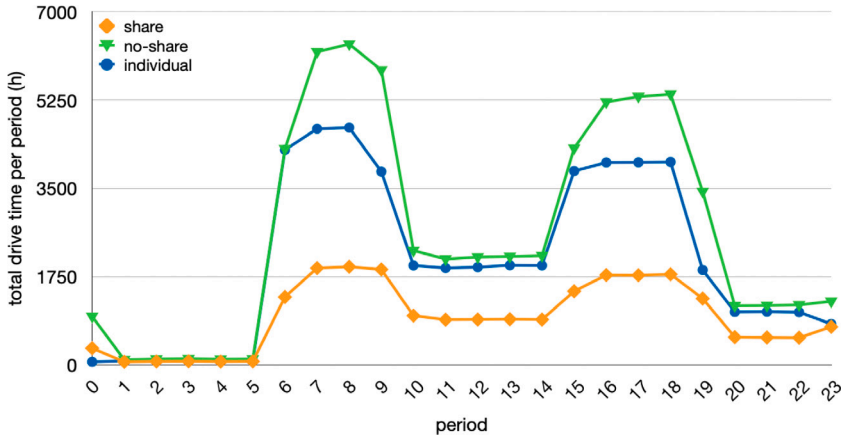


Fig. 7. The vehicle total drive time by period when the number of requests is fixed at 300k.

Table 6
Results with different scenarios when the number of requests is fixed at 300k.

Scenario	δ (min)	α	Q	Fleet size	Average #R per vehicle
0-1	15	2	10	1975	152
1-1	15	2	5	3107	97
1-2	15	2	20	1540	195
2-1	5	2	10	2241	134
2-2	10	2	10	2057	146
2-3	30	2	10	1891	159
3-1	15	1.5	10	2010	149
3-2	15	2.5	10	2062	145

the accumulated drive time of vehicles in each period (excluding period 0, which represents the initial departure of all SAVs from the depot at $t = 0$ to their respective first service points) is reduced thanks to the benefits of ride-sharing and the decrease in the number of en-route vehicles. This effect becomes particularly pronounced during peak hours, which is especially beneficial to limit congestion. In other periods, the benefits of *share* on traffic are more limited, mutualization being mitigated by kilometers traveled empty. Again, for *no-share* configuration, we identify higher drive times compared to others due to too many empty trips.

We now take interest in the occupation situation of vehicles across the day in the system *share*. Given any period h, we keep track of the load of each vehicle at a sampling rate of 1 min. Then, for each vehicle v, we compute a mean load $\bar{q}_{v,h}$ that represents the occupation level of v during h by averaging all the sampled loads of h. To illustrate the system-wide vehicle occupancy during each period h, average, minimum, maximum, and median values of $\bar{q}_{v,h}$ over all vehicles are computed and presented in Fig. 8. We observe that vehicles are generally well occupied during peak hours, primarily driven by typical requests reflecting commuter demands. This surge in peak-hour requests emphasizes the impact of ride-sharing, along with the large-scale effects during these high-demand periods. Furthermore, throughout the day, there are clearly always SAVs actively performing services, the maximum value being as much as the vehicle capacity 10, and SAVs that are basically inactive, the minimum value being 0 during most of the periods. This observation prompts consideration for more efficient dispatching strategies for SAVs, like extending idle times of inactive SAVs and prioritizing insertions into active fleets. As such, other challenges related to autonomous vehicles such as recharging issues may be better addressed in future research.

Finally, we evaluate the sensitivity of the SAV DAR ride-sharing system to different scenarios. Table 6 gives the descriptions of the scenarios tested and the corresponding results. We note that the same request instances are used for all scenarios, aside from the stated changes regarding the length of the time window δ and the maximum ride time amplifier α. Scenario (0-1) corresponds to the basic problem setting. The column *Average #R per vehicle* counts the average number of requests served by an SAV. First of all, we

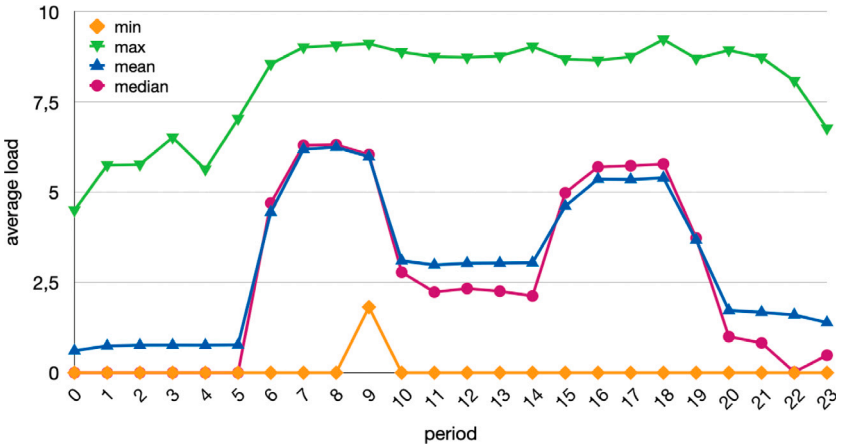


Fig. 8. The occupation situation of SAVs across the day in the SAV DAR ride-sharing system when the number of requests is fixed at 300k.

Table 7
The fleet size under different insertion order strategy.

$\|\mathcal{R}\|$	TWP	STCluster05	STCluster30	STCluster60	Random
10k	**121**	**121**	128	141	177
50k	442	**437**	469	523	708
100k	784	**780**	829	929	1272
200k	1400	**1385**	1489	1672	2251
300k	1988	**1975**	2116	2383	3135

The bold font is for the best result for each problem size.

test the performance of the approach when the vehicle capacity is equal to 5 (scenario (1-1)), 10 (scenario (0-1)), and 20 (scenario (1-2)). We note that the higher the vehicle capacity, the fewer SAVs are used, because more ride-sharing is allowed when the vehicle can accommodate more passengers at one time. Then we try to understand the influence of the request's time constraints on the fleet size, by varying the pickup time window (scenarios (2-1) to (2-3)) and the maximum ride time (scenarios (3-1) to (3-2)). For the constraint on the pickup time window, we conclude without surprise that when it becomes more relaxed, the SAVs are able to serve more requests during the day, and thus fewer vehicles will be needed. While varying the value of the multiplier of the maximum ride time, we observe that α and the fleet size are not necessarily associated. When α is too strict (scenario (3-1)), ride-sharing becomes more difficult; when α is too relaxed (scenario (3-2)), requests can be "unwisely" inserted and affect later insertions.

8.3. Analyses on algorithmic choices on basic resolution of LS_DARP

In this section, we study the effect of different algorithmic choices on the basic resolution configuration of **LS_DARP**, that is, f_0.

Choice of insertion order strategy. Let us start by analyzing the impact on the solutions of different insertion order strategies presented in Section 4.2, namely *TWP*, *Random*, and three variations of *STCluster*: *STCluster05*, *STCluster30*, *STCluster60*. To enhance spatial request clustering, we adopt a broad spatial area divided into 20 disjoint regions. Our default choice *STCluster05* partitions the temporal horizon into time slots of 5 min. And similarly, *STCluster30* and *STCluster60* consider time slots of 30 and 60 min, respectively. Table 7 shows that the processing order of requests plays an important role in the quality of solutions. The classic strategy *TWP* maintains a good performance. Our *STCluster05* outperforms all the other strategies in terms of fleet size. However, not all strategies from the *STCluster* family lead to good results. When the duration of time slots becomes longer, more requests can be classified into the same cluster. Nevertheless, as we deal with one cluster at a time, a large time slot means requests with later departure times can be inserted too early and thus perturb the insertions of earlier requests to be processed in later clusters. The fact that *Random* exhibits much worse solutions emphasizes the significance of following the chronological order of request pickup times. Furthermore, the better performance of *STCluster05* over *TWP* underscores the importance of considering geometric similarity. We note that even though 5 min seem to be short, under the large-scale, high-mutualization context, the number of requests in each cluster obtained from *STCluster05* can sometimes surpass 100 during peak hours. This indicates the effectiveness of this strategy in increasing ride-sharing opportunities.

Choice of route modeling. Now we evaluate the advantages of our *transit network-based* route modeling, compared to the traditional *request-based* method. According to Table 8, the larger the problem size, the greater the reduction in the average route length, which is computed by the number of points in the routes. As the problem size increases, more requests with shared pickup and drop-off locations emerge, allowing aggregation of these requests at the same key point serviced by a single SAV. Furthermore, compared to

Table 8
Comparison between request-based and transit network-based route modelings.

$\|\mathcal{R}\|$	CPU times (min)		Route length	
	Request-based	Network-based	Request-based	Network-based
10k	0.3	0.2 (−33.3%)	176	164 (−6.8%)
50k	12	7.9 (−34.2%)	224	203 (−9.4%)
100k	50.5	34.9 (−30.9%)	258	217 (−15.9%)
200k	207.6	140.9 (−32.1%)	292	241 (−17.5%)
300k	313.5	470.0 (−33.3%)	303	242 (−20.1%)

Table 9
Different filter configurations.

Filter	f_{All}	f_{Auto}	$f_{Auto'}$	f_n	$f_{n'}$	$fRand_n$
ϕ	∞	$g(\mathcal{V}^r)$	$g(\mathcal{V}^r)$	$0.6n$	$0.6n$	/
Regeneration action	/	No	Yes	No	Yes	/

Table 10
General performance with the non-filter configuration f_0.

$\|\mathcal{R}\|$	10k	50k	100k	200k	300k
Fleet size	121	440	780	1385	1975
CPU time (min)	0.2	7.9	34.9	140.9	313.5

traditional modeling, our approach reduces execution times by over 30% for all problem sizes. The decrease in execution times does not vary proportionally with the decrease in route length, mainly because of the additional complexity brought by split insertion resulting from our route modeling.

8.4. Analyses on the effectiveness of the filtering system

We solve the same problems with different variations of filters, and analyze their performances compared to the basic non-filter approach f_0. The spatial–temporal preprocessing of the filtering system divides the transit network into 20 zones with a grid of 4 rows and 5 columns, and divides the time horizon into 24 periods of a length of 1 hour. We recall that the candidate vehicle set $\mathcal{V}^r$ selected by the filtering system is the union of two sets $\mathcal{V}^r_{\oplus}$, where $\oplus$ is the two operators (*sum*, *max*) used to compute the local score $s^{local}_{v,Pr}$.

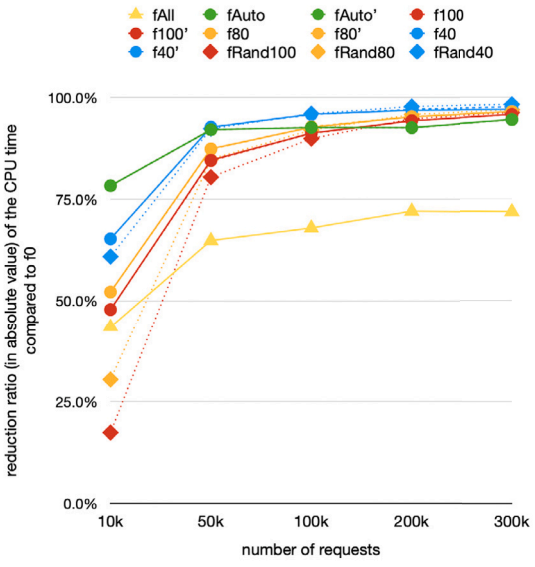
Table 9 summarizes the tested filters with different configurations. The filter f_{All} does not involve the stopping mechanism and explores all candidate vehicles. From empirical experiments, we found that each candidate vehicle causes on average 0.6 calls to the constraint propagation procedure. Then for f_{Auto} and $f_{Auto'}$, ϕ is a variable defined by the function $g(\mathcal{V}^r)$ with

$$g(\mathcal{V}^r) = \begin{cases} 12 & \text{if } |\mathcal{V}^r| \leqslant 200 \\ 0.6 \times 10\% \times |\mathcal{V}^r| & \text{if } 200 < |\mathcal{V}^r| \leqslant 1400 \\ 84 & \text{otherwise} \end{cases},$$

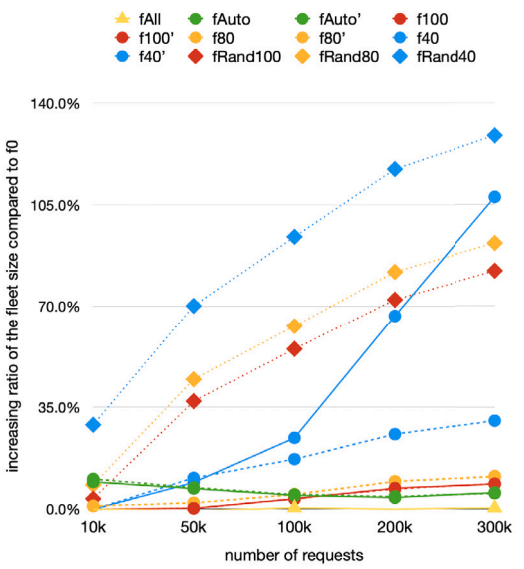
where the lower bound 12 and upper bound 84 indicate that we explore at least 20 and at most 140 candidate vehicles. The filter f_n corresponds to the configuration where the stopping mechanism threshold is $\phi = 0.6n$. In such configuration, we explore around n candidate vehicles from the candidate set $\mathcal{V}^r$. The configuration $f_{n'}$ represents setting ϕ at $0.6n$, and enabling the *regeneration* actions. The values of n considered here are 100, 80 and 40. In addition to configurations induced from the filtering system proposed in this work, we introduce an alternative naive filtering system *fRand* to serve as a reference. Specifically, the variation $fRand_n$ randomly selects n SAVs to be candidate vehicles, and routes of these vehicles are extensively explored.

The general performances of the filters with respect to the final fleet size and the CPU time is presented in Fig. 9. For better illustration, the results are presented as the percentage of variation based on performances of f_0 (see Table 10). The number of requests processed in one day varies from 10,000 to 300,000. Regarding the final fleet size, the gaps between the non-filter approach f_0 and the others using filters become bigger when the system processes more passenger requests. However, most of the gaps remain acceptable except for variations of $fRand$ and f_{40}. The best filter configuration leading to the fewest vehicles is f_{All}, where the candidate SAV set $\mathcal{V}^r$ is extensively explored for each request. In terms of the execution CPU time, when processing a relatively small number of requests (for example, when processing 10,000 requests), the improvements are less important. This is partially because the time saved from the filtering mechanisms is offset by the time needed to update the filtering matrices. When the size of the problem grows, the saved time becomes far more significant, and the advantage of our filtering system becomes more and more obvious.

Table 11 summarizes the experimental results with more details when the number of requests is fixed at 300,000. All the approaches using filters outperform the basic non-filter approach regarding CPU time. We also see a drastic reduction in the number of calls to the propagation of time window constraints (see the column *#updateTW*), which is one of the most time-consuming



(a) The reduction (in absolute value) in CPU times by using different filter configurations



(b) The increment in final fleet sizes by using different filter configurations

Fig. 9. General performances of different filters. Solid lines are filter configurations without regeneration actions, dashed lines are configurations with regeneration actions, dotted lines are $fRand$ configurations.

Table 11
Results with different filters when the number of requests is fixed at 300k.

Filter	CPU time (min)	#updateTW	Fleet size	#regenerations	Total drive time (h)	Passenger average in-vehicle time (min)
f_0	313.5	364,125,925	1975	/	22,937,14	16.64
f_{All}	88.4 (−71.8%)	−8.7%	1983 (+0.4%)	/	+0.2%	−0.1%
f_{Auto}	17.3 (−94.5%)	−82.2%	2087 (+5.7%)	/	+15.2%	−1.1%
$f_{Auto'}$	17.4 (−94.5%)	−82.3%	2082 (+5.4%)	2280	+15.2%	−1.1%
f_{100}	13.4 (−95.7%)	−86.4%	2145 (+8.6%)	/	+18.0%	−1.2%
$f_{100'}$	13.4 (−95.7%)	−86.4%	2147 (+8.7%)	2260	+18.0%	−1.1%
f_{80}	11.2 (−96.4%)	−88.6%	2194 (+11.1%)	/	+20.7%	−1.3%
$f_{80'}$	11.2 (−96.4%)	−88.7%	2198 (+11.3%)	2718	+20.8%	−1.3%
f_{40}	9.4 (−97.0%)	−91.8%	4098 (+107.5%)	/	+56.0%	−3.4%
$f_{40'}$	7.6 (−97.6%)	−92.9%	2576 (+30.5%)	5524	+38.5%	−1.9%
$fRand_{100}$	11.9 (−96.2%)	−91.4%	3595 (+82.0%)	/	+75.1%	−1.7%
$fRand_{80}$	9.7 (−96.9%)	−92.7%	3784 (+91.6%)	/	+83.0%	−2.2%
$fRand_{40}$	5.6 (−98.2%)	−95.5%	4517 (+128.7%)	/	+110.2%	−4.1%

calculation units. When the value of ϕ is properly fixed, the CPU time can be easily reduced by more than 90%. Among all filter configurations that we have tested, the CPU time can be saved by up to 98% (the approaches with $f_{40'}$ and $fRand_{40}$), at the cost of dispatching many more SAVs. As for the final fleet size, all $fRand_n$ filters lead to much worse results than f_n under the same value of n. We also observe that the smaller the value of ϕ, the more vehicles are used. But compared to the CPU time saved, especially for the approaches f_{Auto} and $f_{Auto'}$ that dynamically tune the value of ϕ, the increase in the fleet size seems fair. This further supports our choice of the score for candidate vehicles, whose main objective is to forecast insertion feasibility. Specifically, the results of f_{Auto} demonstrate that selecting the top 10% candidate SAVs with the highest scores is sufficient to ensure a feasible and even favorable insertion. The importance of the regeneration action becomes more evident with small ϕ. For example, it is clear that by setting ϕ to 24 (f_{40}), the final fleet size diverges to an unacceptable value. By implementing the regeneration action ($f_{40'}$), the overall increase in the number of vehicles has been notably curbed. In terms of the total drive time, when ϕ is fixed at a relatively small number, it becomes less satisfying, showing that the candidate score is not quite related to this secondary criterion of solution quality. This raises the challenge of finding a score that takes both quality criteria into account. Finally, it is reasonable that when we use more SAVs to serve the same requests, the quality of service is better in terms of the average in-vehicle time spent by passengers, because there would be less detour with passengers on-board.

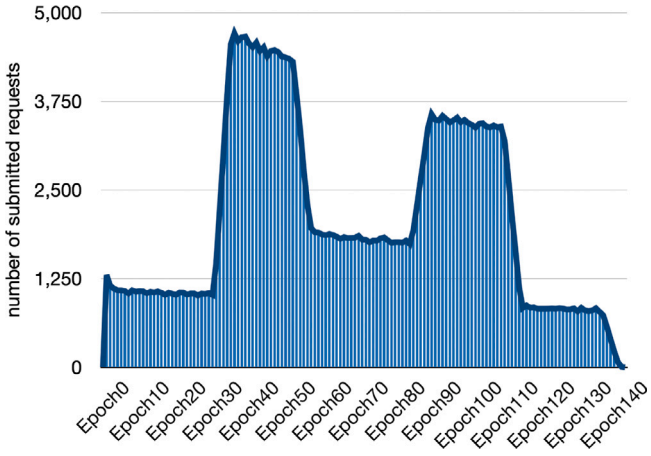


Fig. 10. The number of requests submitted during each epoch.

Table 12
Fleet size under the dynamic context with different filters.

Filter	f_0	f_{All}	f_{Auto}	$f_{Auto'}$	f_{100}	$f_{100'}$	f_{80}	$f_{80'}$	f_{40}	$f_{40'}$
Fleet size	2343	2331	2648	2649	2694	2685	2748	2742	3276	3090
Variation to f_0	–	−0.5%	+13.0%	+13.1%	+15.0%	+14.6%	+17.3%	+17.0%	+39.8%	+31.9%

8.5. Shining lights into a dynamic context

This section is dedicated to exploring the algorithm's adaptation and performance within a straightforward, dynamic context while dealing with 300k requests.

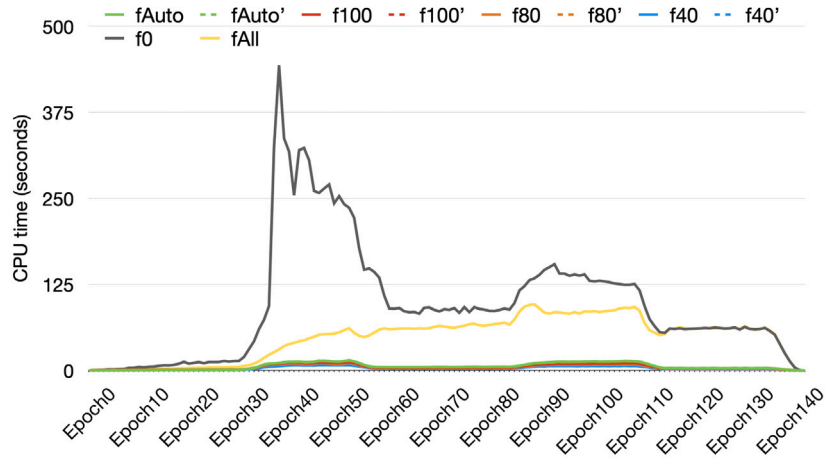
We divide the time horizon into 144 decision epochs of $I_d = 10$ min. At the beginning of each decision epoch, the system *Dynamic_ProcessDARP* deals with requests submitted during the previous epoch. To generate dynamic requests, we keep the same static instances, and randomly set a submission time t^r_{emit} for each request r between 0 and e_{Pr}. Requests are submitted on average one hour before their earliest in-vehicle time. And in line with the no-rejection assumption, values of t^r_{emit} also satisfy that when processing r at epoch, we can always activate a vehicle v currently parking at the depot to serve r. Distributions of requests submitted during each epoch are shown in Fig. 10. High-demand periods during peak hours exhibit two distinct submission peaks.

Solving the problem with different filter configurations, the fleet sizes to be implemented under the underlined context are presented in Table 12. With f_0 being the baseline, we see that f_{All} manages to keep the same (slightly better) level of performance as f_0. For other configurations, higher values of the threshold ϕ correlate with better solution quality.

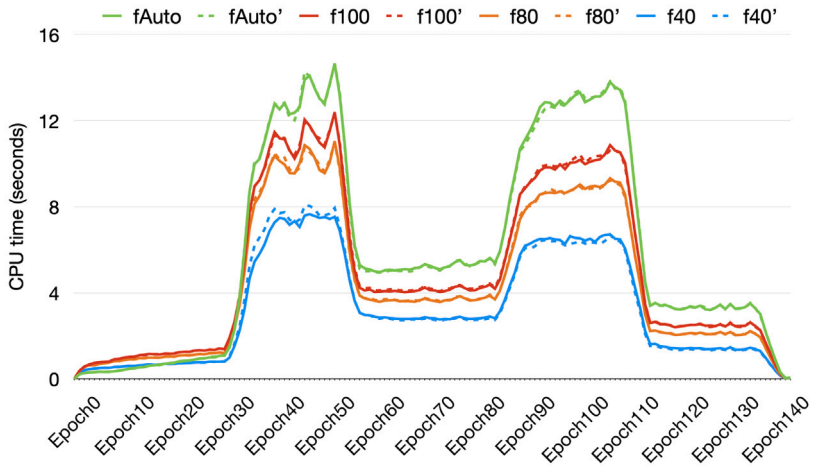
Execution times committed to each decision epoch are shown in Fig. 11. Fig. 11(a) contains results of all configurations, and Fig. 11(b) gives a closer look at results of configurations excluding f_0 and f_{All}. Clearly, f_0 demands notably higher CPU times compared to other configurations, especially during peak hours when the majority of requests are submitted. There are two distinct surges in the figures, corresponding to two peak hours, one in the morning and the other in the afternoon. Notably, the morning surge for f_0 is much higher the afternoon surge. This phenomenon arises from the allowance for early request submissions, leading to a higher volume of pending requests during epochs corresponding to the morning surge. Furthermore, vehicle routes are longer during the morning surge, stemming from inserting requests submitted in advance that have not been executed. Filter f_{All} drastically reduces execution time, particularly during the morning peak. However, we should recall that the dynamic context considered here is relatively simple. In a more realistic scenario, requests submitted significantly in advance of their departure times should undergo re-optimization later, accounting not only for new requests but also stochastic events. This process incurs higher computational costs. As a result, considering potential re-optimization times alongside system updates and communication between SAVs and passengers, the system might necessitate a very short maximum duration μ for *Dynamic_ProcessDARP* to be executed. Consequently, the efficacy of f_{All} can be insufficient, prompting the need to invoke a stopping mechanism. All filter configurations presented in Fig. 11(b) solve the problem of each epoch within 15 s. This demonstrates the great potential of our filtering system when applied in real-life dynamic scenarios.

9. Conclusions

In this paper, we studied a prospective transportation system in which SAVs provide DAR services in urban and rural areas to meet a large number of passenger requests with all kinds of travel needs. We proved with different numerical experiments that such a system helps to effectively lessen the number of en-route vehicles by replacing individual transportation media with medium-sized



(a) The CPU times with all filter configurations



(b) The CPU times without f_0 and f_{All}

Fig. 11. CPU times for each decision epoch using different filters. Solid lines are filter configurations without regeneration actions, dashed lines are configurations with regeneration actions.

SAVs (of capacity 10 in the basic setting). And by enabling ride-sharing, the reduction effect becomes more prominent, gaining sights on the potential of such a system in alleviating traffic pressure and emissions.

In order to solve the resulting large-scale DARP efficiently, we proposed a filtering system. From the macro SAV fleet to the micro insertion positions, different filtering mechanisms are applied sequentially like a cascade to avoid unnecessary and time-consuming calculations. The experimental results showed that, thanks to the filtering system, the computational time can be reduced by almost 97%, while maintaining relatively good performances, especially in terms of the final fleet size.

Moreover, we introduced a new modeling approach to the SAVs' routes based on the real transit network, as well as an adapted greedy insertion algorithm based on it. By aggregating different services at key points, our route with a shorter length significantly reduces the processing time compared to the traditional route modeling method in a large-scale context, and reflects the mutualization of the passengers' trajectories and the traffic conditions. These benefits are supposed to be highly useful in our future work.

We provide related operators with foresight into the SAV fleet size to be fixed in different scenarios by studying the large-scale DARP, and we shed light on the dimension and potential of such a system.

There are still some limitations to this work. For example, some assumptions are not viable for practical applications, such as the one-depot network configuration and the all-homogeneous SAV fleet. Moreover, the question of the underlying network remains open. It is difficult to predict what such a system will be like in the future. Nevertheless, even if technology reaches a tipping point, safe monitoring will almost certainly remain at stake, necessitating some infrastructure (sensors, monitoring devices, specific congestion control, etc.) for the underlying network. Furthermore, even though there are no benchmarks for comparison with such a large-scale SAV DAR system, we believe that the quality of the decision, typically in terms of the total drive time, still has great room for improvement.

There are several directions for future work. Improving the quality of current decisions can be a great challenge. Standard re-optimization methods using linear programming or meta-heuristics may not be applicable due to the large size of our problem, especially when dealing with dynamic requests. However, because of the large size of the request set, which is practically dominated by daily commute demands, we believe that some travel patterns can be extracted, which might help guide making faster and better decisions in more realistic, real-time dynamic scenarios. Furthermore, we have neglected congestion or any other time-dependent issues in this paper, which are of vital significance considering the practical implementation of the system. As a result, this should be another issue addressed in future work. Finally, the specific characteristics of the SAVs can be taken into account during the routing process. For example, with the SAVs being supposed to be electric-powered, it is important to manage the recharging issue during the service period by better dispatching the SAVs, which is also planned to be done in our future work.

CRediT authorship contribution statement

Chijia Liu: Writing – review & editing, Writing – original draft, Visualization, Software, Resources, Methodology, Formal analysis, Data curation, Conceptualization. **Alain Quilliot:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Hélène Toussaint:** Writing – review & editing, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Dominique Feillet:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Acknowledgments

This work was supported by the International Research Center “Innovation Transportation and Production Systems” of the I-SITE CAP 20-25. The authors also express sincere thanks to the reviewers for their valuable insights and constructive feedback, which significantly improved this article.

Appendix A. Time windows on $K(P^r)$ and $K(D^r)$

In this section, we present, when inserting a service X, how the time windows at $K(X)$ are determined.

For $K(P^r)$, we notice that if $n_{P^r} = n_K$, then P^r will be aggregated into K, and $K(P^r) = K$. In this case, the only timestamp that can be influenced is the earliest departure time $e^d_{K(P^r)}$: if the earliest in-vehicle time of r is later than the earliest departing time $e^d_{K(P^r)}$, then v needs to at least wait until e_{P^r} before departing from $K(P^r)$.

$$e^d_{K(P^r)} \leftarrow \max(e_{P^r}, e^d_{K(P^r)}). \tag{17}$$

Otherwise, $K(P^r)$ corresponds to a new key point, whose predecessor and successor are, respectively, denoted as $Pred(K(P^r))$ (which is K) and $Succ(K(P^r))$. The time windows for this new point are determined as follows:

$$e^a_{K(P^r)} = e^d_{Pred(K(P^r))} + t(Pred(K(P^r)), P^r); \tag{18}$$

$$e^d_{K(P^r)} = \max(e_{P^r}, e^a_{K(P^r)}); \tag{19}$$

$$l^d_{K(P^r)} = l^a_{Succ(K(P^r))} - t(P^r, Succ(K(P^r))); \tag{20}$$

$$l^a_{K(P^r)} = l^d_{K(P^r)}. \tag{21}$$

When calculating the lower bounds of time windows of $K(P^r)$, we always begin by $e^a_{K(P^r)}$, then $e^d_{K(P^r)}$: The earliest arrival time at $K(P^r)$ only depends on the earliest departure time from its predecessor $Pred(K(P^r))$ (Eq. (18)); while the earliest departure time from $K(P^r)$ depends on both $e^a_{K(P^r)}$ and the earliest in-vehicle time of request, because the vehicle should wait until passengers get onboard before moving to the next key point (Eq. (19)). As for the upper bounds, we begin by $l^d_{K(P^r)}$ then $l^a_{K(P^r)}$.

For D^r, if the related support key point already exists and coincides with K^d, then only the latest arrival time $l^a_{K(D^r)}$ will be influenced, due to the maximum ride time imposed by r:

$$l^a_{K(D^r)} \leftarrow \min(l^a_{K(D^r)}, \min(l^d_{K(P^r)}, l_{P^r}) + T^r). \tag{22}$$

Otherwise, its time windows are calculated by (23)–(26).

$$e^a_{K(D^r)} = e^d_{Pred(K(D^r))} + t(Pred(K(D^r)), D^r); \tag{23}$$

$$e^d_{K(D^r)} = e^a_{K(D^r)}; \tag{24}$$

$$l^d_{K(D^r)} = l^a_{Succ(K(D^r))} - t(D^r, Succ(K(D^r))); \tag{25}$$

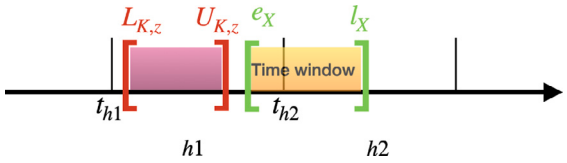


Fig. 12. There is no intersection between the possible passage interval on z_X for K and the time window imposed by X, therefore, an insertion after K is not considered.

$$l^a_{K(D^r)} = \min(l^d_{K(D^r)}, \min(l^d_{K(P^r)}, l_{P^r}) + T^r). \tag{26}$$

We note that the vehicle's latest arrival time at $K(D^r)$ is determined by the maximum ride time constraint imposed by r. The term $\min(l^d_{K(P^r)}, l_{P^r})$ in Eqs. (22) and (26) calculates the latest time for r to get onboard from its pickup location. If l_{P^r} is earlier than $l^d_{K(P^r)}$, passengers of r are allowed to board at l_{P^r} and wait until the vehicle is ready to depart, as long as their maximum ride time limit is not exceeded, which also explains why in Eqs. (20) and (21), the vehicle's latest departure and arrival on $K(P^r)$ are not explicitly related to l_{P^r}.

Appendix B. A relaxed test on the insertion positions

We assume in this paper that traffic condition is invariant and that we have a pre-computed travel-time matrix. However, we are convinced that an effective DAR system must be able to accommodate dynamic, time-dependent contexts. Our algorithmic approaches are intended to be flexible enough to adapt to such scenarios. To achieve this adaptability, we adopt a strategy of postponing the utilization of travel times $t(\cdot,\cdot)$ to the latest possible stage. When transitioning to time-dependent contexts, this method eliminates the requirement for computationally intensive, initial calculations of travel times.

In addition to what we have introduced in the filtering system, a *relaxed test* can be introduced in the *InsertionPositionFilter1* procedure of the insertion position filtering module, before proceeding with the tests (7) to (14):

For the relaxed test, the filtering matrix M^v is used. For each key point K in one of the cells $M^v[z_X, h]$ for $h \in \mathcal{H}_X$, we check the intersection between:

- The time window $[e_X, l_X]$ defined for X.
- The time interval $[L_{K,z}, U_{K,z}]$ during which it is estimated that v can pass through z departing from K. Note that this interval is not kept in M^v but can be easily deduced from the information on elastic duration.

If the intersection is not empty, the key point passes the test, and we continue with the strict test. Otherwise, as illustrated in Fig. 12, we consider that X cannot be inserted after the key point K. This test can be performed in $O(1)$.

Appendix C. Routine proof of insertion feasibility check

In Section 6.4, we have claimed that (11), (12) and (9) can, respectively, be seen as necessary conditions of (7), (8) and (13). The proofs for the necessity between (11) and (7), and between (12) and (8) are notably evident; we entrust their verification to the readers for their own exploration.

For the necessity between conditions (9) and (13) can be proved as follows:

Necessity between conditions (9) and (13). On the one hand, we have

$$l^a_K \leqslant l^a_{Succ(K)} - t(K, Succ(K)), \tag{27}$$

on the other hand, according to the triangle inequality, we have:

$$t(X, K) + t(K, Succ(K)) \geqslant t(X, Succ(K)). \tag{28}$$

Therefore, if Constraint (13) is satisfied, thanks to the relation (27), we have:

$$e_X + t(X, K) \leqslant l^a_{Succ(K)} - t(K, Succ(K)), \tag{29}$$

then

$$e_X + t(X, K) + t(K, Succ(K)) \leqslant l^a_{Succ(K)}. \tag{30}$$

Finally, thanks to the relation (28), we end up with Constraint (9). □

References

Agatz, Niels, Erera, Alan, Savelsbergh, Martin, Wang, Xing, 2012. Optimization for dynamic ride-sharing: A review. European J. Oper. Res. 223 (2), 295–303.

Alonso-Mora, Javier, Samaranayake, Samitha, Wallar, Alex, Frazzoli, Emilio, Rus, Daniela, 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. Proc. Natl. Acad. Sci. 114 (3), 462–467.

Bertsimas, Dimitris, Jaillet, Patrick, Martin, Sébastien, 2019. Online vehicle routing: the edge of optimization in large-scale applications. Oper. Res. 67 (1), 143–162.

Bongiovanni, Claudia, Kaspi, Mor, Geroliminis, Nikolas, 2019. The electric autonomous dial-a-ride problem. Transp. Res. B 122, 436–456.

Calvo, Roberto Wolfler, Colorni, Alberto, 2007. An effective and fast heuristic for the dial-a-ride problem. 4OR 5 (1), 61–73.

Cordeau, Jean-François, 2006. A branch-and-cut algorithm for the dial-a-ride problem. Oper. Res. 54 (3), 573–586, Publisher: INFORMS.

Cordeau, Jean-François, Laporte, Gilbert, 2003. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transp. Res. B 37 (6), 579–594.

Diana, Marco, Dessouky, Maged M., 2004. A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. Transp. Res. B 38 (6), 539–557.

Fagnant, Daniel J., Kockelman, Kara M., 2018. Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas. Transportation 45 (1), 143–158.

Furuhata, Masabumi, Dessouky, Maged, Ordóñez, Fernando, Brunet, Marc-Etienne, Wang, Xiaoqing, Koenig, Sven, 2013. Ridesharing: The state-of-the-art and future directions. Transp. Res. B 57, 28–46.

Gavalas, Damianos, Konstantopoulos, Charalampos, Pantziou, Grammati E., 2015. Design and management of vehicle sharing systems: A survey of algorithmic approaches. CoRR, abs/1510.01158.

Gendreau, Michel, Tarantilis, Christos D., 2010. Solving Large-Scale Vehicle Routing Problems with Time Windows: The State-of-the-Art. Cirrelt Montreal.

Ho, Sin C., Szeto, W.Y., Kuo, Yong-Hong, Leung, Janny M.Y., Petering, Matthew, Tou, Terence W.H., 2018. A survey of dial-a-ride problems: Literature review and recent developments. Transp. Res. B 111, 395–421.

Horn, Mark E.T., 2002. Fleet scheduling and dispatching for demand-responsive passenger services. Transp. Res. C 10 (1), 35–63.

Hunsaker, Brady, Savelsbergh, Martin W.P., 2002. Efficient feasibility testing for dial-a-ride problems. Oper. Res. Lett. 30, 169–173.

Jain, Siddhartha, Van Hentenryck, Pascal, 2011. Large neighborhood search for dial-a-ride problems. In: Lee, Jimmy (Ed.), Principles and Practice of Constraint Programming – CP 2011. In: Lecture Notes in Computer Science, vol. 6876, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 400–413.

Kullman, Nicholas D, Cousineau, Martin, Goodson, Justin C, Mendoza, Jorge E., 2021. Dynamic ridehailing with electric vehicles. Transp. Sci..

Levin, Michael W., 2017. Congestion-aware system optimal route choice for shared autonomous vehicles. Transp. Res. C 82, 229–247.

Levin, Michael W., Kockelman, Kara M., Boyles, Stephen D., Li, Tianxin, 2017. A general framework for modeling shared autonomous vehicles with dynamic network-loading and dynamic ride-sharing application. Comput. Environ. Urban Syst. 64, 373–383.

Liang, Xiao, Correia, Gonçalo Homem de Almeida, An, Kun, van Arem, Bart, 2020. Automated taxis' dial-a-ride problem with ride-sharing considering congestion-based dynamic travel times. Transp. Res. C 112, 260–281.

Maghraoui, Ouaïl Al, Vosooghi, Reza, Mourad, Abood, Kamel, Joseph, Puchinger, Jakob, Vallet, Flore, Yannou, Bernard, 2020. Shared autonomous vehicle services and user taste variations: Survey and model applications. Transp. Res. Procedia 47, 3–10.

Merat, Natasha, Madigan, Ruth, Nordhoff, Sina, 2017. Human Factors, User Requirements, and User Acceptance of Ride-Sharing in Automated Vehicles. Technical Report, OCDE, Paris.

Molenbruch, Yves, Braekers, Kris, Caris, An, 2017. Typology and literature review for dial-a-ride problems. Ann. Oper. Res. 259, 295–325.

Nansubuga, Brenda, Kowalkowski, Christian, 2021. Carsharing: a systematic literature review and research agenda. J. Serv. Manag. 32 (6), 55–91, Publisher: Emerald Publishing Limited.

Narayanan, Santhanakrishnan, Chaniotakis, Emmanouil, Antoniou, Constantinos, 2020. Shared autonomous vehicle services: A comprehensive review. Transp. Res. C 111, 255–293.

Nazari, MohammadReza, Oroojlooy, Afshin, Snyder, Lawrence V., Takác, Martin, 2018. Deep reinforcement learning for solving the vehicle routing problem. CoRR, abs/1802.04240.

Riley, Connor, Van Hentenryck, Pascal, Yuan, Enpeng, 2020. Real-time dispatching of large-scale ride-sharing systems: Integrating Optimization, Machine Learning, and Model predictive control. arXiv:2003.10942 [cs, math].

Rist, Yannik, Forbes, Michael, 2021. A new formulation for the dial-a-ride problem. Transp. Sci. 55.

Rossi, Federico, Zhang, Rick, Hindy, Yousef, Pavone, Marco, 2018. Routing autonomous vehicles in congested transportation networks: structural properties and coordination algorithms. Auton. Robots 42 (7), 1427–1442.

Schulz, Arne, Pfeiffer, Christian, 2024. Using fixed paths to improve branch-and-cut algorithms for precedence-constrained routing problems. European J. Oper. Res. 312 (2), 456–472.

Shuo Ma, Yu Zheng, Wolfson, O., 2013. T-share: A large-scale dynamic taxi ridesharing service. In: 2013 IEEE 29th International Conference on Data Engineering. ICDE, IEEE, Brisbane, QLD, pp. 410–421.

Tafreshian, Amirmahdi, Abdolmaleki, Mojtaba, Masoud, Neda, Wang, Huizhu, 2021. Proactive shuttle dispatching in large-scale dynamic dial-a-ride systems. Transp. Res. B 150, 227–259.

Tang, Jiafu, Kong, Yuan, Lau, Henry, Ip, Andrew W.H., 2010. A note on "efficient feasibility testing for dial-a-ride problems". Oper. Res. Lett. 38 (5), 405–407.

Varone, Sacha, Janilionis, Vytenis, 2014. Insertion heuristic for a dynamic dial-a-ride problem using geographical maps. In: Proceedings of the 10th International Conference on Modeling, Optimization and Simulation, Nancy, France. p. 8.