https://doi.org/10.31449/inf.v49i1.6756 Informatica **49** (2025) 19–38 **19**

Dynamic Routing for Large-Scale Mobility On-Demand Transportation Systems

Chijia Liu[1], Alain Quilliot[1], Hélène Toussaint[1], Dominique Feillet[2]
[1]Université Clermont Auvergne, CNRS, UMR 6158 LIMOS, 63178 Aubière, France
[2]Mines Saint-Etienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, F - 42023 Saint-Etienne, France
E-mail: chijia.liu@uca.fr, alain.quilliot@uca.fr, helene.toussaint@uca.fr,feillet@emse.fr

Keywords: Large-scale dial-a-ride problem, ride-sharing, shared autonomous vehicles

Received: July 23, 2024

We study a prospective large-scale Ride-Sharing Mobility-on-Demand (RSMoD) transportation system. In this system, a fleet of Shared Autonomous Vehicles (SAVs) provides services to a very large number of passengers (up to 300,000). Passenger requests are submitted throughout the service horizon and require an immediate response from the system. We formulate the resulting decision problem as a dynamic dial-a-ride problem (DARP), characterized by its very large size, necessitating well-fitted filtering devices.
We first derive a static version of this dynamic DARP from a statistical knowledge of the requests. Solving this static DARP version allows us to identify reference requests and related routes, which we incorporate into a Guided Insertion Mechanism (GIM). This mechanism aims to expedite the insertion of real dynamic requests. To handle requests that do not fit the GIM learning basis, we complement this mechanism with a Filtering System (FS), creating an algorithmic GIM-FS framework that dynamically routes the SAVs and assigns them to requests. Numerical tests focus on the behavior of both this prospective system RSMoD and the GIM-FS algorithmic scheme. They show that a large-scale RSMoD system involving SAVs is likely to significantly reduce the number of on-route vehicles as well as the energy consumption and that the two-phase GIM-FS approach is more efficient than a baseline method that does not learn from a pre-processed static version of the DARP

Povzetek: Raziskava obravnava dinamično usmerjanje v velikih sistemih prevoza na zahtevo z avtonomnimi vozili. Predlagan je dvofazni algoritem, ki vključuje mehanizem vodenega vstavljanja in filtrirni sistem za učinkovito dodeljevanje vozil zahtevam. Numerični testi kažejo na zmanjšanje števila vozil na cesti in manjšo porabe energije.

1 Introduction

The concept of *Mobility-on-Demand* (MoD) has gained a reputation for being accessible and available to users whenever needed, thus offering more flexible transportation services [4, 40, 29]. On-demand ride services are usually provided by *Transportation Network Companies* (TNCs) such as Uber and Lyft, which communicate with users through digital platforms and mobile applications. Some of these on-demand ride services implement *ride-sharing* or *ride-pooling* [2, 42], allowing passengers with similar itineraries to share vehicles, thus alleviating both carbon emissions and traffic pressure. We refer to these services as *Ride Sharing Mobility-on-Demand* (RSMoD) systems.

Nowadays, most MoD or RSMoD systems aim to provide convenient transportation for residents living in areas with sparse public transportation, such as suburban or rural areas, and for elderly and disabled passengers [36, 35]. These systems are small due to the costs of drivers. However, the current development of autonomous car technology suggests that integrating autonomous vehicles into RSMoD systems could overcome these limitations [46, 16, 37].

Connected autonomous vehicles would drastically decrease the costs of RSMoD systems and make them accessible to all kinds of users. In urban areas, their flexibility might allow them to occupy a significant market share between standard public transportation systems (bus, tram, and underground) and individual cars, whose excessive number and energy consumption are currently major concerns for public authorities. In most European cities, public authorities are promoting both the emergence of RSMoD systems and advances in autonomous vehicle technology [28, 30], with the prospect that a significant portion of individual vehicles might soon be replaced by *Shared Autonomous Vehicles* (SAVs) involved in large-scale RSMoD systems.

In this study, we address some of the decision problems likely to be related to the management of a prospective RSMoD system, relying on SAVs and thus capable of capturing a large part of individual mobility in a medium-sized urban/peri-urban area, such as the one surrounding

the town of Clermont-Ferrand in France (population around 300,000). We consider a fleet of mid-sized SAVs controlled by a service-providing company to meet a very large number of daily passenger requests (around 300,000, i.e., around 40% of the individual mobility currently handled by private cars). Installing and managing such a fleet requires handling several decision levels: strategic ones, such as pricing the services, determining the size of the fleet, and establishing the infrastructure of the support transit network (monitoring devices, sensors, protected areas, etc.); operational ones, such as vehicle routing and scheduling, assigning vehicles to requests, maintenance scheduling, etc. In our case, we focus on two tightly connected questions:

- The size of the fleet, which may be considered a decision at the strategic level, which also determines at the operational level the number of requests likely to be rejected.
- The operational dynamic (online) assignment of the large number of requests to the vehicles, and the dynamic routing and scheduling of the vehicles.

While addressing these two questions, we follow a two-fold purpose: Designing efficient algorithms for the real-time management of large sets of requests and vehicles, and the minimization of both the running costs and the number of unsatisfied requests; observing the behavior of the RSMoD system itself, including its size, costs, and its potential positive impacts on urban mobility.

To achieve this purpose, we presuppose the availability of a suitable statistical representation of the mobility demand. This initial statistical knowledge allows us to solve a virtual static version of our dynamic *dial-a-ride problem* (DARP) and use the resulting virtual collection of routes as a learning basis for fast decision-making. Our main contributions are as follows:

- We design a two-phase algorithm that dynamically inserts the requests into the routes followed by the vehicles while minimizing the running costs. This algorithm relies on:
  - A *Guided Insertion Mechanism* (GIM) that takes advantage of the large-scale feature of the system and works as a learning device. It derives high-quality reference routes from the resolution of a static virtual version of the DARP, and uses them to insert real requests r into real vehicles v;
  - A *Filtering System* (FS), that applies to a given request r in case the GIM process fails to efficiently assign a vehicle to r. FS reduces the search space by filtering the candidate insertion parameters, namely the vehicles and their insertion positions within the vehicle routes.
- We conduct numerical experiments that assert the ability of this algorithm to significantly reduce computational times compared to the baseline approach (which does not involve the GIM mechanism), while maintaining the quality of the resulting solution.
- Furthermore, according to experimental results, an RSMoD system relying on SAVs might reduce the number of on-route vehicles by around 99% compared to a private system where all 300,000 requests would be serviced by private vehicles. These experiments also reveal that such an RSMoD system would manage significantly more dynamic requests in a shorter total drive time than a taxi-like MoD system involving the same number of vehicles and lacking a ride-sharing feature

To our knowledge, this is the first study that explicitly links the resolution of dynamic DARP to the use of statistical vehicle travel patterns.

The paper is organized as follows: Section 2 is devoted to the state of the art. Section 3 describes how our prospective RSMoD system works. We present in Section 4 a static version of the resulting large-scale DARP and describe how it can be handled through a best-fit insertion heuristic. The core of our contribution is in Section 5, which describes the two-phase GIM-FS algorithm for the handling of the dynamic version of the large-scale DARP. Section 6 presents numerical experiments. We briefly conclude with a discussion about open questions.

2 Related works

2.1 MoD systems

Together with the significant increase in smartphone ownership and the growing demand for flexible mobility, *Mobility-on-Demand* (MoD) transportation systems have emerged in recent years, greatly facilitating travel for urban and rural residents. MoD systems manage requests through digital platforms to customize efficient travel routes. Since the use of *Shared Autonomous Vehicles* (SAVs) decreases the running costs of MoD systems and opens the way to large-scale MoD systems, academics are increasingly interested in studying these systems [13, 1]. Research in this area generally falls into the following two categories.

The first category contains studies focusing on the systems themselves, including their different configurations, deployment viability, and potential advantages. For example, in [15], the authors consider a dynamic RSMoD system, where a fleet of SAVs is employed to cater for $54,324$ passenger requests during a service horizon of 24 hours in the city of Austin. They demonstrate the ability of the ride-sharing feature to reduce SAVs' total travel time. In [26], the author focuses on the congestion issue. He sets up a Linear Programming model and runs experiments that reveal that, due to the empty repositioning trips performed by the SAVs, congestion during peak hours may increase. In [28], the authors introduce a simulation framework to analyze the behavior of an MoD system with Electric SAVs

(SAEVs). They show that optimizing the number of charging stations and reducing charging times are crucial for the efficiency of the SAEV fleet.

The second category contains studies on how requests and vehicles might be efficiently managed. It puts algorithm design at the core of the research: passenger requests are submitted online, hoping for almost immediate feedback. Deciding on the requests means efficiently assigning them to vehicles and accordingly routing and scheduling the vehicles. This defines the dynamic version of the *Dial-a-Ride* Problem (DARP) (see [38]). The case when the number of requests is high (large-scale MoD systems) raises the issue of filtering, concerning the introduction of filtering devices that will guide the decision-makers, for any given request, towards the best-fitted vehicles. For example, authors in [43] introduce a *dual-sided taxi searching* approach to efficiently filter candidate vehicles inside a taxi system that dynamically handles over 330,000 requests. In [3], the authors implement a similar idea to match requests and vehicles in a ride-sharing system with thousands of requests.

2.2 Static DARP

Addressing the dynamic version of the problem requires the ability to efficiently solve its static version, which is characterized by the fact that all requests are known in advance and there are no restrictions regarding computational times. Reviewing the literature reveals that most studies on the DARP have been conducted concerning its static version [12].

The static DARP involves a set of passenger requests, each typically containing a pick-up location (origin) and a drop-off location (destination). Additionally, certain constraints regarding service quality are imposed, such as time windows for pick-up or drop-off services, or a maximum ride time. The goal is to organize the routes of a fleet of vehicles so that certain objectives are optimized, such as minimizing total travel time or maximizing service quality. Meanwhile, routes should satisfy constraints such as vehicle capacity, maximum route duration, and those imposed by the requests [18].

For very small-sized DARPs, authors propose exact algorithms. Studies [38, 39] introduce dynamic programming algorithms to solve specific instances of DARPs. Exact algorithms based on Branch-and-Bound, Branch-and-Cut, and Branch-and-Price [10, 7] are the most common methods used to solve the DARP. For example, in [32], the authors propose a Brand-and-Price method to solve a DARP with *Split Requests and Profit* (DARPSRP) with an objective of maximizing the total profit. Requests involving several passengers can be split to be served separately if it is of benefit to do so. Different instances with up to 4 vehicles and 40 requests, or 5 vehicles and 20 requests can be solved to optimality.

In real-life situations, the number of requests to be processed usually forbids exact methods and leads to the design of heuristics. In [11], authors design a tabu search heuristic to solve the DARP with time windows and deal with instances with up to 295 requests with 20 vehicles. Solving the largest instances necessitates over 200 minutes of CPU time. Their work has inspired other tabu search-based methods [6, 31]. Other types of metaheuristics are also widely studied, such as Variable Neighborhood Search (VNS) [33], Large Neighborhood Search (LNS) [21, 34] and Adaptive Large Neighborhood Search (ALNS) [47, 5].

Insertion heuristics [22], which gradually constructs feasible solutions, may be adapted in a natural way to dynamic contexts and mid-to-large-sized problems [8, 41, 20]. Among them, the *best-fit insertion* heuristics [44] appear as the most efficient ones. In order to deal with very large instances, [27] introduces a filtering system based on a spatial-temporal decomposition, which, given a target request, not only selects candidate vehicles but also provides the candidate insertion positions within the route of these vehicles where the request can be inserted. Part of this filtering system will be used in this paper.

2.3 Learning travel patterns

In large-scale systems, most requests correspond to travelers who make similar trips from one day to another, and thus adhere to some statistical patterns that one may try to capture through statistical analysis or machine learning [17, 49]. For example, in [48], the authors propose a deep learning-based approach, *ST-ResNet*, which models the temporal closeness and properties of crowd traffic and predicts the inflow and outflow of crowds in every region. In [45], authors introduce a learning model based on a convolution network to predict hourly Origin-Destination Matrices in the Beijing urban area.

On the other hand, due to the similarity in passenger requests, the trajectories of vehicles supporting these requests are also likely to exhibit some repetitive or similar travel patterns [25, 19]. In [24], the authors construct a sequence of dynamic graphs from the aggregation of trajectory data and apply graph mining algorithms to analyze the spatial and temporal travel patterns in the transit network. In [23], a trajectory clustering method based on identifying the *Longest Common Sequence* of each trajectory is presented, which facilitates traffic flow pattern analysis.

3 A target prospective RSMoD system

The transit network The key underlying infrastructure of our system is a transit network, represented here as an oriented graph $\mathcal{G} = (\mathcal{N}^\dagger, \mathcal{A})$, where $\mathcal{A}$ is the set of road links (elementary arcs) and $\mathcal{N}^\dagger$ is the set of road junctions (nodes). We denote by $\mathcal{N} \subset \mathcal{N}^\dagger$ the nodes where a vehicle involved in the system may pick up or drop some passengers. We suppose, for the sake of simplicity, that the travel time on any arc $a \in \mathcal{A}$ for any vehicle v is constant and that the travel times from any node $n_1 \in \mathcal{N}^\dagger$ to any node

$n_2 \in \mathcal{N}^\dagger$ are pre-calculated. We denote by $t(n_1, n_2)$ this quantity. In addition, we only consider one vehicle depot $n_0 \in \mathcal{N}^\dagger$ in the network. Throughout this paper, we refer to this network as representative of the transit network of a mid-sized urban area with about 500,000 inhabitants, involving between 5,000 and 15,000 nodes.

The main players Three agents should be identified as the main players of our RSMoD system: the central operator, the SAV fleet, and the passengers. Passengers submit their travel requests online to the central operator via a digital terminal. The central operator then assigns requests to SAVs and plans the routes for these vehicles. Finally, each SAV follows its route to provide services to passengers. More precisely:

– **The Passenger Requests**: A passenger request r, submitted at time $tSub^r$ is defined by: a load (number of passengers) q^r; a pick-up service O^r, containing a pick-up node $o^r \in \mathcal{N}$ and a pick-up time window $[e_{o^r}, l_{o^r}]$, where e_{o^r} and l_{o^r} represent respectively the earliest and latest pick-up time for r; a drop-off service D^r, containing a drop-off location $d^r \in \mathcal{N}$, and a maximum ride-time T^r. We easily deduce a drop-off time window $[e_{d^r}, l_{d^r}]$, where

$$e_{d^r} = e_{o^r} + t(o^r, d^r);$$

and

$$l_{d^r} = l_{o^r} + T^r.$$

We assume that the service times at pick-up and drop-off locations can be neglected.

The Large Scale Feature: We assume that this prospective RSMoD system is likely to replace around 40% of the current mobility currently reliant on individual cars. So we deal here with a number of requests between 2.10^5 and 4.10^5.

Statistical Behavior of the Requests: Requests are submitted online throughout the time horizon and they may differ from one day to another. However, we suppose that they follow some probabilistic distribution that can be reproduced through simulation.

– **The SAV Vehicles**: The SAV fleet involves V identical autonomous electric vehicles, all with capacity Q. The running times of the vehicles along the arcs of the transit network come with the network. Each request can be served by at most one vehicle, and no transfers between vehicles are allowed. Once again, we do not address the energy issue and assume that the vehicles are able to run without recharging throughout the time horizon.

– **The Central Operator**: The central operator is the *decider* of the system. It collects the requests and assigns them to the vehicles (or possibly rejects them), while simultaneously routing, scheduling, and controlling the vehicles. It processes the requests by "packages". More precisely, $[0, T]$ being the planning horizon, it divides it into E intervals $[t_e, t_e + I_E]$ of duration I_E, resulting in a set of *decision epochs* $\mathcal{E} = \{0, 1, \ldots, e, \ldots, E-1\}$. At each decision epoch $e \in \mathcal{E}$, it collects a set $\mathcal{R}^{e+1}$ of requests that will be processed during epoch $e+1$ according to the following decision scheme ($\mathcal{R}^0$ means the request that have been submitted in advance) :

Step 1 At the beginning of epoch e, the central operator proceeds the requests of $\mathcal{R}^e$ during $[t_e, t_e+\eta]$, where $\eta < I_E$ is a parameter that bounds the computational time allowed to this part of the process. We note that η must be small enough to leave enough time for time-consuming steps 2 and 3.

Step 2 Then it informs all passengers about the way their requests have been handled and asks them to confirm.

Step 3 According to this it communicates with the vehicles and updates their routes and tentative schedules.

The **primary objective** of the central operator is to minimize the number of *rejected* requests, that cannot be serviced by the vehicle fleet. The **secondary objective** is the minimization of an operational cost, reduced here to the Vehicles' total Travel Time, denoted by VTT.

Remark 1: In this work, we care neither about stochastic events (passenger no-shows, traffic accidents, etc.), nor about the communications processes that take place between the passengers, the central operator and the vehicles, nor, as previously mentioned, about the energetic issue. Yet it will be indispensable to address issues while designing the processes aiming at the control of any real-life *RSMoD* system involving SAVs.

Remark 2: It is worth noting that treating each decision epoch as having equal duration I_E is a simplification. In real-life dynamic systems, the frequency of initiating routing and replanning procedures usually varies according to how quickly passengers should receive a response and the quality of the routing. However, this aspect is not the main concern of this study.

Our Target Problem According to the above description, our challenge becomes the design of algorithmic tools for the handling, at any epoch e, of the requests r of $\mathcal{R}^e$. In order to deal with this dynamic large-scale DARP, we first solve a *static* (where all requests are known in advance) *virtual* version of our DARP. Next, we use the virtual solution obtained this way as a reference assignment and routing strategy, and handle the real requests while trying to mimic this strategy. In case we fail to insert a request r

in the current collection Θ of vehicle routes, we rely on a complementary filtering device to compute the well-fitted insertion parameters of the insertion of r into Θ.

4 Dealing With a virtual static large-scale dial-a-ride problem

Formally, the main components of the *static* DARP are adapted from what has just been previously described:

- The Transit Network.
- The Vehicle Fleet: Vehicles are the same as described above, but their number is not known in advance. Consequently, this Static DARP also aims to help decide the size of the vehicle fleet. In addition, it could potentially be utilized to identify the part of the transit network that should support some kind of monitoring infrastructure. Nevertheless, this specific issue is not addressed in this study.
- The requests: We consider a large-scale set $\mathcal{R}$ of known-in-advance *virtual* requests. They are generated according to the probabilistic distribution of real requests. Being static, these requests are not characterized by submission times.

The goal is to build a collection $\Theta = \{\theta^v, v = 1, \ldots, V\}$ of routes, assign every request to some route, and schedule the vehicles along those routes following the objective below:

- (Primary Objective) Minimize V.
- (Secondary Objective) Minimize VTT.

This performance criterion is based on a lexicographic ordering of the pairs (V, VTT): (V_1, VTT_1) is better than (V_2, VTT_2), if and only if:

- $V_1 < V_2$, or
- $V_1 = V_2$ and $VTT_1 < VTT_2$.

Notice that our two criteria are in some way antagonistic. We denote the resulting problem by *S_DARP*.

4.1 Encoding a route: notion of key point

The large scale of our problem leads us to rely on a non-standard representation of the routes: For any vehicle v, its route θ^v is a list $\theta^v = \{P_0, \ldots, P_i, \ldots, P_{M(v)-1}\}$ of *key points*, where a *key point* is defined by all the drop-off and pick-up services to be simultaneously performed by v at a specific node $n_P \in \mathcal{N}$. Proceeding this way will drastically diminish the length of the routes with respect to the standard encoding and make it easier to perform both the search of well-fitted insertion parameters for the requests into the vehicles and the feasibility tests related to such an insertion. More specifically, each key point $P \in \theta^v$ specifies: (a) a service location $n_P \in \mathcal{N}$; (b) an arrival time window $[e_P^a, l_P^a]$, where e_P^a and l_P^a indicate the earliest and latest possible arrival times at node n_P, respectively, together with the set R_P^- of requests scheduled to get out of the vehicle at P (drop-off services). (c) a departure time window $[e_P^d, l_P^d]$, where e_P^d and l_P^d represent the earliest and latest departure times from n_P respectively, together with the set R_P^+ of requests scheduled to get in (pick up services) the vehicle at P; (d) the load of vehicle q_P before leaving n_P.

The first and last points in θ^v, i.e., P_0 and P_{M-1}, represent the fact that every vehicle should leave the depot to start providing service, and return to the depot after having finished all services by the end of time horizon T. We also denote them by $P_0 = D_1$ and $P_{M-1} = D_2$. Every route is initialized as $\{D_1, D_2\}$, with: $n_{D_1} = n_{D_2} = n_0$, $e_{D_1}^a = e_{D_1}^d = e_{D_2}^a = e_{D_2}^d = 0$, $l_{D_1}^a = l_{D_1}^d = l_{D_2}^a = l_{D_2}^d = T$, $R_{D_1}^+ = R_{D_1}^- = R_{D_2}^+ = R_{D_2}^- = \varnothing$, and $q_{D_1} = q_{D_2} = 0$. For the sake of simplicity, given two key points $P_i \in \theta^v$ and $P_j \in \theta^v$, we use $t(P_i, P_j)$ to denote the travel time from node n_{P_i} to node n_{P_j}. For any request r, we denote by $P(O^r)$ and $P(D^r)$ the key points supporting O^r and D^r, respectively.

Travel Time and Feasibility of a Route

According to this specific encoding, the travel time of a route $\theta^v = \{P_0, \ldots, P_i, \ldots, P_{M-1}\}$ is equal to $\sum_{i \leq M-2} t(P_i, P_{i+1})$. Besides, θ^v is *feasible* if it satisfies the following constraints:

- For any request r inserted at θ^v, the pick up service O^r should take place before the drop-off service D^r: if $P(O^r) = P_i$ and $P(D^r) = P_j$, then $i < j$;
- For any i, the load inside the vehicle must not exceed its capacity:

$$q_{P_i} \leqslant Q; \tag{1}$$

- For any i, v should not leave P_i before arriving at it:

$$e_{P_i}^a \leqslant e_{P_i}^d, \tag{2}$$

$$l_{P_i}^a \leqslant l_{P_i}^d; \tag{3}$$

- For any $i < M-1$, the arrival time at P_{i+1} should be consistent with the departure time from P_i:

$$e_{P_i}^d + t(P_i, P_{i+1}) \leqslant e_{P_{i+1}}^a, \tag{4}$$

$$l_{P_i}^d + t(P_i, P_{i+1}) \leqslant l_{P_{i+1}}^a; \tag{5}$$

- For any request r inserted at θ^v, its maximum ride-time should be bounded:

$$e_{P(D^r)}^a - \min(e_{P(O^r)}^d, l_{o^r}) \leqslant T^r, \tag{6}$$

$$l_{P(D^r)}^a - \min(l_{P(O^r)}^d, l_{o^r}) \leqslant T^r, \tag{7}$$

where the term $\min(e^d_{P(O^r)}, l_{o^r})$ represents the actual earliest in-vehicle time of r. Generally, passengers are expected to board the vehicle no earlier than the earliest departure time from O^r, denoted $e^d_{P(O^r)}$. However, if $e^d_{P(O^r)}$ is later than r's latest permissible in-vehicle time, l_{o^r}, then r may board at l_{o^r} and wait until the vehicle is ready. This situation could arise, for example, if the vehicle must wait for other requests to get onboard at the same key point before it can depart. The same reasoning applies to how we represent the actual latest pick-up time of r, $\min(l^d_{P(O^r)}, l_{o^r})$.

– For any r getting in v at P_i, v must be able to arrive at n_{P_i} before the latest pick-up time of r:

$$\forall r \in \mathcal{R}^+_{P_i},\ e^a_{P_i} \leqslant l_{o^r}; \tag{8}$$

– For any i, no related time windows are empty:

$$e^a_{P_i} \leqslant l^a_{P_i}, \tag{9}$$

$$e^d_{P_i} \leqslant l^d_{P_i}. \tag{10}$$

4.2 A best-fit insertion heuristic

In the static context, we are given *a-priori* unlimited time to solve *S_DARP*. Yet both the problem's very large size and its complexity forbid exact methods. So we handle *S_DARP* through a *greedy Best-Fit insertion heuristic* (BF). This greedy heuristic might be augmented with local search devices and meta-heuristic schemes. However, it is not the focus of this study, and the performance of the *best-fit* insertion heuristic is enough concerning our purpose [20].

4.2.1 The insertion procedures

Since we rely here on a specific encoding of the routes, we need to explain the way we adapt the standard insertion procedures to this encoding: Inserting a request r in a route θ^v according to insertion parameters (v, P_o, P_d) usually means inserting O^r between P_o and its successor in θ^v and D^r between P_d and its successor in θ^v. Here, the fact that a key point refers to several pick-up and drop-off services leads us to consider several cases (for the sake of simplicity, we only consider O^r and P_o):

1. If $o^r \neq n_{P_o}$, then inserting O^r means creating a new key point. There are two ways to insert this new key point into θ^v. The first one (*standard* one, with a parameter $\epsilon_o = 0$) means inserting it between P_o and its successor in θ^v; The second one (*split* one, with a parameter $\epsilon_o = 1$) means splitting the current key point P_o into 2 key points P^{arr}_o and P^{dep}_o, that respectively represents the drop-off and pick-up services performed by the vehicle at n_{P_o}, and inserting the new key point between P^{arr}_o and P^{dep}_o. In practice, this *split* insertion means that v arrives at n_{P_o}, performs its drop-off service, moves to O^r to pick up r and finally comes back to n_{P_o} to perform the pick-up part of its service.
2. If $o^r = n_{P_o}$, then no additional key point is created, and the pick-up service O^r is aggregated to the pick-up service related to P_o, whose time window is impacted.

Consequently, the insertion parameters of the insertion of r into current route collection Θ define a 5-tuple $(v, P_o, \epsilon_o, P_d, \epsilon_d)$.

4.2.2 A best-fit insertion heuristic

According to this, we adapt the classic greedy insertion heuristic framework presented in [22]:

Step 1: Sort $\mathcal{R}$ according to the *ST-Cluster strategy*: Divide the transit network into small zones and the time horizon $[0, T]$ into small periods and cluster the requests according to their pick-up and drop-off zones and their earliest pick-up time periods (see [27] for details). Then handle the clusters according to increasing periods, while randomly selecting the requests belonging to the same cluster.

Step 2: For each request $r \in \mathcal{R}$, sorted according to Step 1:

Step 2.1: For any *insertion parameter* 5-tuple $(v, P_o, \epsilon_o, P_d, \epsilon_d)$, where $v \in \mathcal{V}$ is an activated vehicle, P_o located before P_d in θ^v, check the feasibility of the insertion of O^r according to P_o and ϵ_o, and D^r according to P_d and ϵ_d.

Step 2.2: If at least one insertion is feasible, keep the *best-fitted* one that minimizes detour made by v, and insert r; otherwise, activate a new vehicle v and insert r into the related trivial route.

4.3 Performing step 2.1: checking the feasibility of an insertion

We briefly provide here some insight into Step 2.1. For the sake of simplicity, we restrict ourselves to the case $(\epsilon_o = 0, \epsilon_d = 0)$, which means to the case of *standard* insertions.

4.3.1 A cascade strategy

We proceed according to a *cascade* strategy: We first perform a set of *fast* tests. In case of success, we continue to perform a more time-consuming constraint propagation process that checks the mathematical feasibility of the target insertion. This cascade strategy comes as follows:

– First, we test the insertion feasibility of O^r:

  – If $o^r = n_{P_o}$, then O^r is directly aggregated at P_o, and $P(O^r) = P_o$. We increase the load q_{P_o} by q^r and add r into the set $\mathcal{R}^+_{P_o}$. In addition, we update the earliest departure time $e^d_{P_o}$:

$$e^d_{P_o} \leftarrow \max(e^d_{P_o}, e_{o^r}). \tag{11}$$

The other timestamps of the passage time windows are not directly impacted.

- If $o^r \neq n_{P_o}$, then $P(O^r)$ is a new key point to be inserted between P_o and P_{o+1}, with $n_{P(O^r)} = o^r$, $\mathcal{R}^+_{P(O^r)} = \{r\}$ and $q_{P(O^r)} = q_{P_o} + q^r$. The earliest arrival time and the latest departure time of $P(O^r)$ are computed:

$$e^a_{P(O^r)} = e^d_{P_o} + t(P_o, P(O^r)) \quad (12)$$

$$l^d_{P(O^r)} = l^a_{P_{o+1}} - t(P(O^r), P_{o+1}) \quad (13)$$

- Then, we test the insertion feasibility of D^r:
  - Proceed with D^r and P_d as with O^r and P_o;
  - Deal with the maximum ride time T^r as follows: Two key points $P(O^r)$ and $P(D^r)$ are now associated with O^r and D^r. Depending on the case, they have just been created or they were previously existing. In any case, we may denote by $[e^d_{P(O^r)}, l^d_{P(O^r)}]$ the departure time window of $P(O^r)$ and by $[e^a_{P(D^r)}, l^a_{P(D^r)}]$ the arrival time window of $P(D^r)$. Then we update these time windows:

$$e^d_{P(O^r)} \leftarrow \max(e^d_{P(O^r)}, e^a_{P(D^r)} - T^r) \quad (14)$$

$$l^a_{P(D^r)} \leftarrow \min(l^a_{P(D^r)}, l^d_{P(O^r)} + T^r) \quad (15)$$

  - Increase by q^r the load of all key points between $P(O^r)$ and $P(D^r)$ and check that the capacity constraint (1) is not violated.
  - Apply the *constraint propagation procedure* and accordingly adjust the time windows along θ^v.

4.3.2 The constraint propagation procedure

The *constraint propagation procedure* [14, 27], checks the consistency of current time windows along the route θ^v with the temporal constraints induced by the *S_DARP* and accordingly updates the influenced time windows. This procedure is the most time-consuming component of our *cascade* strategy, with a complexity $O(M^2)$, where M is the number of key points in θ^v.

Remind that a route is feasible if constraints (2) to (10) in Section 4.1 are satisfied. Constraint propagation means that the violation of such a constraint will trigger a *constraint propagation rule*:

- If (2) is violated, then $e^d_{P_i} \leftarrow e^a_{P_i}$;
  if (3) is violated, then $l^a_{P_i} \leftarrow l^d_{P_i}$.
- If (4) is violated, then $e^a_{P_{i+1}} \leftarrow e^a_{P_i} + t(P_i, P_{i+1})$;
  if (5) is violated, then $l^d_{P_{i-1}} \leftarrow l^a_{P_i} - t(P_{i-1}, P_i)$.
- If (6) is violated, then for any $r \in \mathcal{R}^-_{P_i}$, delay the earliest departing time at its destination: $e^d_{P(O^r)} \leftarrow e^a_{P_i} - T^r$;
  if (7) is violated, then for any $r \in \mathcal{R}^+_{P_i}$, decrease the latest arrival time at its destination: $l^a_{P(D^r)} \leftarrow l^d_{P_i} + T^r$.
- If any of constraints (8) to (10) is violated, then return a *Fail* signal.

Starting from the initial triggers induced by the first updates at $P(O^r)$ and $P(D^r)$ ((11) to (15)), the procedure propagates the temporal *S_DARP* constraints all along θ^v while updating the related time windows according to the above rules. The procedure ends when no new trigger requires activation or when a *Fail* signal is emitted. In such a case, the current insertion parameter 5-tuple $(v, P_o, \epsilon_o, P_d, \epsilon_d)$ is discarded as infeasible.

5 Dealing with the original dynamic large-scale dial-a-ride problem

We may now come back to our original dynamic large-scale DARP as defined in Section 3. We denote the problem by D_DARP. We suppose here that the fleet size V is fixed, and that the requests are dynamically submitted to the system along the time horizon $[0, T]$, which is divided into E decision epochs. Remind that $\{\mathcal{R}^e,\ e \in \mathcal{E}\}$ denotes the set of requests submitted during epoch $e-1$. It is not known in advance and differs from one day to another. As in *S_DARP*, our goal is to assign the requests to the vehicles and route these vehicles. But our lexicographic performance criterion becomes:

- (Primary Objective) Minimizing the number of rejected requests.
- (Secondary Objective) Minimizing VTT.

We must take into account the *dynamic* constraint that keeps us from processing a request before it is submitted. As mentioned in Section 3, we apply the following decision framework:

1. Initialize the route collection Θ by processing the set R^0 of requests that have been submitted before the beginning of the process. During all the decision processes, we use the same key point-based route encoding as in Section 4.
2. For $e = 0, \ldots, E-1$ do
   (a) ***D_DARP***$(\mathcal{R}^e)$ **instruction**: Process the requests $r \in \mathcal{R}^e$ during the time interval $[t_e, t_e + \eta]$;
   (b) **Dispatch** the decision towards the vehicles and the users during the time interval $[t_e + \eta, t_{e+1}]$. Update the current state of the system while making the vehicles achieve their duty at epoch e while following the earliest possible time according to current time windows.

The rest of this section is devoted to the algorithmic part of this decision framework, i.e., the *D_DARP*$(\mathcal{R}^e)$ instruction.

5.1 Two-phase algorithmic scheme for *D_DARP*($\mathcal{R}^e$)

To address the dynamic version of the DARP, we first employ *preprocessing* that leverages our statistical knowledge of the request distribution and the resolution to the static problem introduced in Section 4. The preprocessing helps to obtain an estimation of the optimal fleet size, a collection of reference (virtual) requests, and a collection of *travel patterns*. These elements form the basis of the *Guided Insertion Mechanism*, which will be discussed later. Then we perform the *D_DARP*($\mathcal{R}^e$) instruction by applying the following two-phase framework, termed *GIM-FS*:

- **Phase 1: Guided Insertion Mechanism** Each request $r \in \mathcal{R}^e$ is processed by a *Guided Insertion Mechanism* (GIM), which tries almost immediate insertions of r into the current route collection Θ under the guidance of insertion patterns deduced from the reference requests and preprocessing. If the insertion of r is feasible according to GIM, then we keep the best-fitted one and insert r;
- **Phase 2: Filtering System** For any request r not inserted by GIM, we invoke a specific *Filtering System* (FS) to fast identify the best candidate insertion parameters. If there are parameters that allow a feasible insertion of r then we choose the best-fitted one; otherwise, we reject r.

Let us now describe the preprocessing process, which serves as the basis for GIM, before delving into the details of both GIM and FS.

5.2 Preprocessing

Urban short-distance trips primarily consist of commuting, shopping, and other personal errands, making up the majority of daily travel purposes [9]. Therefore, large sets of requests should present some similar patterns. Consequently, if all requests are served by a *RSMoD* system, daily vehicle travel patterns should also be characterized by some similar patterns. GIM is based on the same idea that drives statistical learning: if two large request sets $\bar{\mathcal{R}}$ and $\mathcal{R}$ are similar from a statistical point of view, then vehicle routes $\bar{\Theta}$ designed for $\bar{\mathcal{R}}$ and vehicle routes Θ designed for $\mathcal{R}$ should also be similar.

5.2.1 Solving reference static problem

According to the above motivation, we start by addressing a *reference* static problem $S\_DARP(\bar{\mathcal{R}})$ as introduced in Section 4, where $\bar{\mathcal{R}}$ is a *virtual reference* request set generated so that it is consistent with our statistical knowledge of the requests. As much time as necessary is used to solve $S\_DARP(\bar{\mathcal{R}})$ to get a good *reference* route set $\bar{\Theta} = \{\bar{\theta}^v, v \in 1, \ldots, V\}$. At this time, a point needs to be discussed. We do not know *a-priori* the fleet size V required to efficiently handle our original problem $D\_DARP(\mathcal{R})$. Experience shows that the way requests' submission times evolve from one day to another is far more volatile than the other characteristics of the requests. Therefore, we intuitively guess that, for the same number of requests, the *online* feature will make the fleet size larger than that obtained through the resolution of the corresponding static DARP. Consequently, we consider the size of $\bar{\mathcal{R}}$ as a parameter and generate $\bar{\mathcal{R}}$ that reproduces the spatio-temporal distribution of the real requests, so that the resulting fleet size V is close to the optimal size required by the real system. This requires some tuning processes that will be discussed in Section 6.2 presenting numerical experiments.

5.2.2 Deriving travel patterns

Next, from the reference route set $\bar{\Theta}$, we derive a set of *travel patterns* Γ. Specifically, for each vehicle $v \in \mathcal{V}$, a *travel pattern* $\gamma^v \in \Gamma$ can be derived from its reference route $\bar{\theta}^v = \{\bar{P}_0, \ldots, \bar{P}_{M-1}\}$. We define the *travel pattern* γ^v as a list of *guiding points*, where each guiding point $G \in \gamma^v$ is a representation of a *key point cluster* in $\bar{\theta}^v$.

Definition 5.1 (*Key Point Cluster*)**.** A subroute $\{P_j, P_{j+1}, \ldots, P_{j+m}\}$ of route $\theta^v = \{P_0, \ldots, P_i, \ldots, P_{M-1}\}$ with M key points forms a key point cluster, if and only if:

$$t(P_k, P_{k+1}) \leqslant \alpha, \text{ for } j \leqslant k \leqslant j+m-1$$

$$t(P_{j-1}, P_j) > \alpha, \text{ if } j \neq 0$$

$$t(P_{j+m}, P_{j+m+1}) > \alpha, \text{ if } j+m \neq M-1$$

where the parameter α is a temporal threshold defining the maximum travel time (distance) allowed for two consecutive key points belonging to the same cluster.

A guiding point G representing the cluster $[\bar{P}_j, \bar{P}_{j+1}, \ldots, \bar{P}_{j+m}]$ is associated with a temporal neighborhood $[t_G^{min}, t_G^{max}]$, with $t_G^{min} = \min_{\bar{P} \in G} e^a_{\bar{P}}$, and $t_G^{max} = \max_{\bar{P} \in G} l^d_{\bar{P}}$. We note that $\bar{D}_1$ and $\bar{D}_2$, which correspond to the initial and terminal reference key points $\bar{P}_0$ and $\bar{P}_{M-1}$ respectively, are excluded from the travel pattern construction because no insertion happens at these points. For any reference key point $\bar{P}$ other than $\bar{D}_1$ and $\bar{D}_2$, its associated guiding point is denoted by $G(\bar{P})$.

As shown in Figure 1, given the reference route $\bar{\theta}^v$, the corresponding travel pattern is γ^v. $\{\bar{P}_2, \bar{P}_3, \bar{P}_4\}$, $\{\bar{P}_6, \bar{P}_7, \bar{P}_8, \bar{P}_9\}$ and $\{\bar{P}_{11}, \bar{P}_{12}\}$ are three key point clusters containing multiple key points. For example, the associated guiding point of $\bar{P}_3$ is $G(\bar{P}_3) = G_1$.

5.3 Phase 1: the guided insertion mechanism

The idea of GIM when solving the real dynamic problem is to encourage every SAV v to follow its pre-defined travel pattern γ^v. Let us consider a request $r \in \mathcal{R}^e$ to be inserted. In the following, we describe how GIM proceeds r.

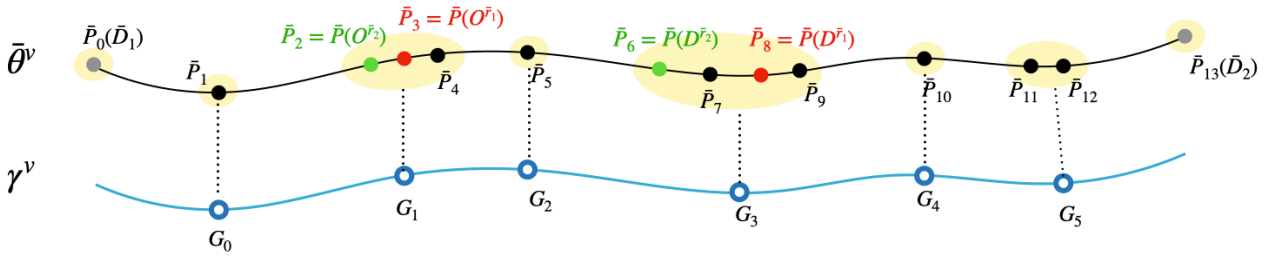


Figure 1: Reference route $\bar{\theta}^v$ and travel pattern γ^v

First of all, we retrieve a list of reference requests from $\bar{\mathcal{R}}$ which are *similar* to r. We say that a reference request $\bar{r}$ is similar to r if and only if:

$$\max(t(O^r, O^{\bar{r}}), t(O^{\bar{r}}, O^r)) \leqslant \delta^s$$

$$\max(t(D^r, D^{\bar{r}}), t(D^{\bar{r}}, D^r)) \leqslant \delta^s$$

and

$$|e_{o^r} - e_{o^{\bar{r}}}| \leqslant \delta^t$$

where parameter δ^s and δ^t are *similarity* parameters: δ^s bounds the travel time (distance) between the origins (resp. destinations) of two similar requests, and δ^t bounds the difference between the earliest pick-up times between r and $\bar{r}$. Let $\bar{\mathcal{R}}^r$ denote the set of reference requests similar to r. In order to speed the construction of $\bar{\mathcal{R}}^r$, we divide the transit networks into *zones* and the time horizon into *periods* (which are typically larger than what have been utilized in the *ST-Cluster* strategy). For every request r (reference or real), we quickly identify the zone $z(O^r)$ (resp. $z(D^r)$) related to O^r (resp. D^r) and the periods $H(O^r)$ (resp.$H(D^r)$) during which O^r (resp. D^r) can be serviced. Then we restrict the search for similar reference requests to those in $\bar{\mathcal{R}}$ whose origin and destination lie in $z(O^r)$ and $z(D^r)$ respectively, and whose time windows are consistent with $H(O^r)$ and $H(D^r)$.

Next, we retrieve a set of *guiding objects* $GO^r = \{\dots, (\gamma^v, G^o, G^d), \dots\}$ from $\bar{\mathcal{R}}^r$. If two reference requests are inserted in the same reference route and their insertion positions belong to the same key point clusters, then they correspond to the same guiding object. In such a case, we only keep one occurrence of each guiding object in GO^r. For example, in figure 1, if $\bar{r}_1$ and $\bar{r}_2$ are both similar to the target real request r, then only one occurrence of (γ^v, G_2, G_4) is kept in GO^r.

Once GO^r is constructed, we retrieve the candidate insertion parameters for r. Specifically, a key point P in a real route θ^v is called a *child* of a guiding point $G \in \gamma^v$, if and only if the time windows of P overlap with the temporal neighborhood of G, that means if:

$$e_P^a \leqslant t_G^{max}$$

and

$$l_P^d \geqslant t_G^{min}.$$

We notice that a guiding point $G \in \gamma^v$ may have multiple *children* in θ^v, these children being consecutive. Symmetrically, a key point can also have multiple *parents* in γ^v. For example, in Figure 2, P_2, P_3 and P_4 are children of G_2 and P_4 has two parents G_2 and G_3. For each guiding point G, we only memorize lc_G, the *predecessor* of the *left-most child* of G, and rc_G, the *right-most child* of G. In Figure 2 we have $lc_{G_1} = P_1$ and $rc_{G_1} = P_4$. If D_2 is the only key point satisfying the above condition then we set $rc_G = lc_G$. At the beginning, all real routes only consist of the two special key points D_1 and D_2. For every $G \in \gamma^v$, both lc_G and rc_G are initialized to be D_1. Then, given $(\gamma^v, G^o, G^d) \in GO^r$, the corresponding candidate insertion parameters are 5-tuples $(v, P_o, \epsilon_o, P_d, \epsilon_d)$ such that $P_o \in \mathcal{L}^o$ and $P_d \in \mathcal{L}^d$, where $\mathcal{L}^o$ is the list of candidate insertion positions for O^r, containing all the key points between lc_{G^o} (included) and rc_{G^o} (included), the same definition holding for $\mathcal{L}^d$. For the guiding object (γ^v, G^o, G^d) of Figure 2, we have $\mathcal{L}^o = \{P_1, P_2, P_3, P_4\}$ and $\mathcal{L}^d = \{P_7\}$.

Finally, the *best-fit insertion heuristic* is utilized over all the candidate insertion parameters given by GO^r. In other words, among all feasible insertion parameters, we keep the one that minimizes the insertion cost, and insert r for real. However, if r fails to be inserted via GIM, we put it aside and try the next request in $\mathcal{R}^e$. We wait until Phase 2 before coming back to r. Every time a request is inserted into v via GIM, time windows of key points along θ^v are modified, and lc_G and rc_G are updated for $G \in \gamma^v$.

5.4 Phase 2: the filtering system

At each decision epoch e, Phase 2 of the *GIM-FS* framework consists of trying to insert the requests $r \in \mathcal{R}^e$ rejected by GIM in Phase 1, while keeping with the best-fit insertion principle. Yet, the current state of the system may involve thousands of vehicles and hundreds of thousands of key points. Consequently, the number of potential insertion parameters $(v, P_o, \epsilon_o, P_d, \epsilon_d)$ may be too large regarding the computational time η allowed to execute the assignment/routing process. So we must filter the search for the insertion parameters. For that, we adapt the *Filtering System* (FS) introduced in [27] to the dynamic context. The trick is to maintain throughout the decision

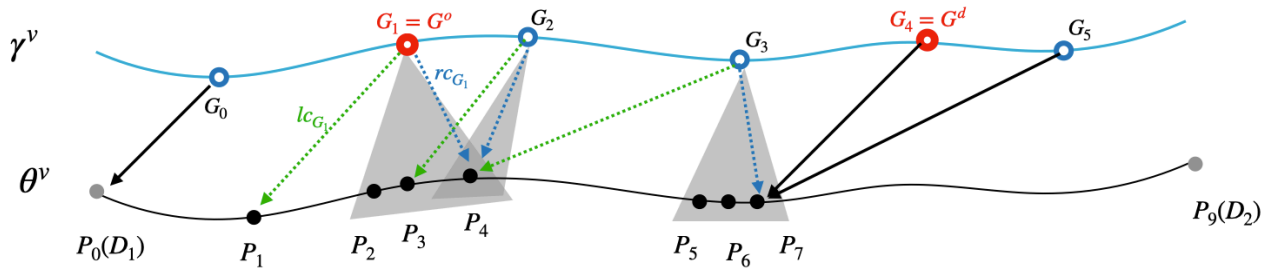


Figure 2: The projection between a travel pattern γ^v and the real route θ^v. For a point G, the dashed green arrow points at lc_G, the dashed green arrow points at rc_G, and the solid black arrow means that $lc_G = rc_G$. For the sake of illustration, we select several guiding points and use emitting gray shadows to elaborate their children

process a skeleton representation of the routes that involves a partition of the transit networks into *zones* and of the time horizon into *periods*. These periods and zones are the same as in Section 5.3 and allow us to quickly identify the vehicles and the key points likely to lead to feasible insertions of a given request. In the following, we briefly present FS.

As mentioned above, the transit network space is partitioned into a set of *zones*, $\mathcal{Z}$ and the time horizon is partitioned into a set of *periods*, $\mathcal{H}$, whose granularity may be adapted according to the context. Then the skeleton representation of the current state of the system is provided by a collection of *filtering* matrices: the *vehicle filtering matrix* $M^{\oplus}[\mathcal{Z} \times \mathcal{H}]$, where $\oplus$ is an operator to be clarified later, and the *insertion position filtering matrices* $M^v[\mathcal{Z} \times \mathcal{H}], v = 1, \ldots, V$. The matrices are defined as follows.

- For any zone z, any period h and any key point P related to vehicle v, we define the *elastic duration* $El_{P,z,h}$ as $El_{P,z,h} = \min(ub_{P,z}, t_{h+1}) - \max(lb_{P,z}, t_h)$, where $lb_{P,z}$ (resp. $ub_{P,z}$) denotes the estimated earliest (resp. latest) time when v can arrive at (resp. depart from) at z while coming from (resp. going to) P. These two values are computed using the *four-references estimator* [27] which determines the estimated travel time between any node and any zone. Intuitively, $El_{P,z,h}$ provides us with an estimation of the *plausibility* that v may enter into z during period h while moving from P. The larger the value of $El_{P,z,h}$, the more plausible the move becomes.
- Then, for any zone z and any period h, the *vehicle filtering* matrix $M^{\oplus}$ identifies vehicles that are able to move through zone z during period h, together with the key points likely to allow this incursion. More precisely, we denote by $Pl^{\oplus}_{v,z,h}$ the quantity obtained by applying the associate operator $\oplus$ to the elastic durations $El_{P,z,h}$, $P \in \theta^v$. $Pl^{\oplus}_{v,z,h}$ provides us with the *plausibility* of the presence of vehicle v inside zone z during period h. The operator $\oplus$ may correspond to either the sum or the max of its arguments, or to the sum of the two largest elements. According to this, $M^{\oplus}[z,h]$ is defined as the set of the vehicles v such that $Pl^{\oplus}_{v,z,h}$ is not null:

$$M^{\oplus}[z,h] = \{(v, Pl^{\oplus}_{v,z,h}) | v \in \mathcal{V}, Pl^{\oplus}_{v,z,h} > 0\}$$

- For each vehicle v, the insertion position filtering matrix M^v contains tuples $(P, El_{P,z,h})$ for each $P \in \theta^v$ and z, h such that $El_{P,z,h} > 0$. Such a matrix may be understood as a reverse matrix of $M^{\oplus}$ and helps us in updating $M^{\oplus}$ every time an insertion is performed.

Identification of Candidate Vehicles: When we try to insert a new request r, we first utilize the vehicle filtering matrices $M^{\oplus}$ to quickly identify a subset of candidate vehicles $\mathcal{V}^r$. We denote by $z(O^r) \in \mathcal{Z}$ (resp. $z(D^r)$) the zone related to O^r (resp. D^r) and by $H(O^r) \subset \mathcal{H}$ (resp. $H(D^r)$) the periods when O^r (resp. D^r) may be serviced. The vehicle candidate set $\mathcal{V}^{r,\oplus}$ contains vehicles that appear in both $M^{\oplus}[z(O^r), H(O^r)]$ and $M^{\oplus}[z(D^r), H(D^r)]$.

Identification of Candidate Insertion Parameters: For each candidate v, potential insertion positions are determined from the matrix M^v. A key point P_o is regarded as a candidate insertion position for O^r if it appears in $M^v[z(O^r), H(O^r)]$ and meets conditions ensuring that independently inserting O^r at P does not violate load and time constraints on P_o. The same validation is applied for D^r. The candidate positions then undergo the complete feasibility test explained in Section 4.1, and the best-fitted one is retained. If no viable insertion is found, r is rejected.

Stopping mechanism A *stopping mechanism* can be integrated into FS to further accelerate the process. It relies on a trial threshold $\mathcal{T}$ and a counter τ, that is initialized to 0 every time we start inserting a request r and that is incremented every time some insertion parameter $(v, P_o, \epsilon_o, P_d, \epsilon_d)$ is tried for insertion. The increment consists in an estimation of the computational effort induced by this trial. In practice, this computational effort is set to 1 if the *constraint propagation procedure* is called and to 0 else. Then the candidate vehicles v in $\mathcal{V}^r$ are sorted according to a score, that combines $Pl^{\oplus}_{v,z(O^r),h \in H(O^r)}$ and

$Pl^{\oplus}_{v,z(D^r),h\in H(D^r)}$ in order to quantify the plausibility that v may service both the origin and destination of r. The search for insertion parameters $(v, P_o, \epsilon_o, P_d, \epsilon_d)$ proceeds by scanning the candidate vehicles v according to decreasing score values. It stops when counter τ reaches the threshold value $\mathcal{T}$.

Therefore, FS has two variations. If the stopping mechanism is not implemented, it is referred to as *Partial Filtering System* (PFS); otherwise, we denote it as *Complete Filtering System* (CFS).

6 Numerical experiments

We implemented all the tested algorithms in the C++ language and executed the computations on a machine equipped with an AMD EPYC 7452 32-Core Processor and 512 GB of RAM.

6.1 Instances

Transit network We consider the transit network in Clermont-Ferrand, a mid-sized city in France, and its suburban area. The underlined graph is downloaded from OpenStreetMap, containing $31,357$ arcs and $13,839$ nodes, among which $1,496$ are selected as pick-up and drop-off points. Each pick-up and drop-off point is associated with a label *working*, *residential*, or *undefined*. In the underlined graph, 6.7% of the points are of type *working*, 54% of type *residential*, and 39.3% of type *undefined* (see Figure 3).

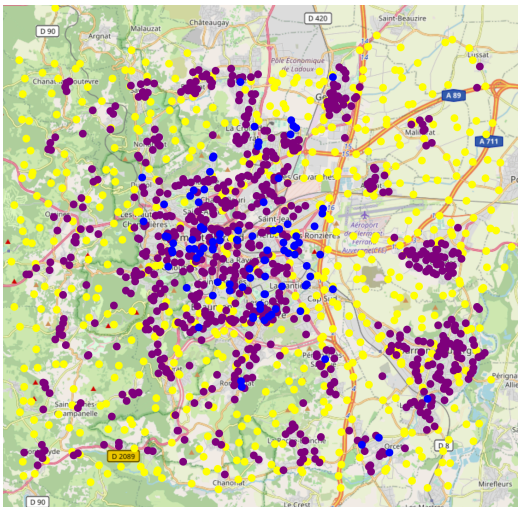


Figure 3: The pick-up and drop-off locations in the underlined transit network in Clermont-Ferrand. Blue, purple and yellow points represent points of type *working*, *residential*, and *undefined*, respectively

The vehicles The capacity of the vehicles is equal to 10. Determining the fleet size is part of the problem (remind that the request rejection rate and VTT are antagonistic criteria) and will be discussed in the next section.

Time horizon and decision epochs The *RSMoD* service period lasts for $T = 24$ hours, starting at 00:00 and ending at 24:00. Requests are collected and processed every 10 minutes. Therefore, within a time horizon of $T = 24$ hours, we end up with 144 decision epochs with $I_E = 10$ minutes. We use η as a parameter for sensitivity analysis.

Request distribution and configuration The large-scale *RSMoD* system studied here being still prospective, no suitable data is available to perfectly capture the usage of such a system. We have designed a passenger request generator to generate daily request instances that comply with the intended use case of the system. For each day, we consider five time slots: **Morning Slack (MS)**, from 00:00 to 06:00; **Morning Peak (MP)**, from 06:00 to 10:00; **Normal Hours (NH)**, from 10:00 to 15:00; **Evening Peak (EP)**, from 15:00 to 19:00; and **Evening Slack (ES)**, from 19:00 to 24:00. Then we ensure that 1% of the daily requests depart during **MS**, 20% during **NH**, 9% during **ES**, while **MP** and **EP** each accounts for 35% of the requests. The two peak periods correspond to passenger commuting trends. As for the spatial distribution of the requests, around 70% originate from *residential* (resp. *working*) points and terminate at *working* (resp. *residential*) points. They are referred to as "typical" passenger requests. Around 80% of "typical" requests traveling from o to d during **MP** (resp. **EP**) are related to a "symmetrical" passenger request traveling from d back to o during **EP** (resp. **MP**).

Each request r has a pick-up time window δ^r between 10 and 30 minutes, and a maximum ride-time $T^r = 2 \times t(o^r, d^r)$, where $t(o^r, d^r)$ denotes the fastest travel time from its origin to its destination. As for the submission times, 90% of the requests are *urgent* ones, with a difference between the submission time and the earliest pick-up time smaller than 1.5 hours. The rest are *relaxed* requests, that are submitted much earlier (at least 1.5 hours before their earliest pick-up time) or in advance. If we divide the time horizon $[0, T]$ into time bins of equal duration of 10 minutes, the distribution of requests submitted at each time bin is given in Figure 4.

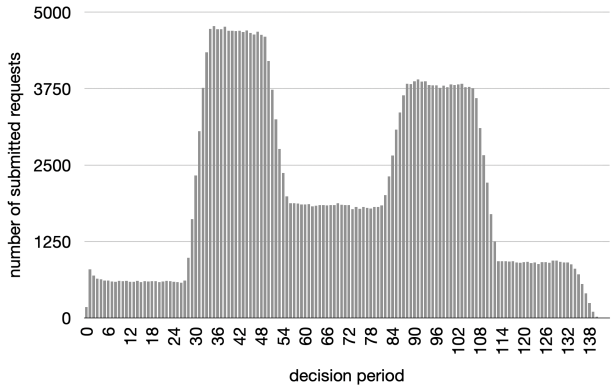


Figure 4: The number of requests submitted at each time bin

Recurrent Requests: We introduce a proportion β of $\mathcal{R}$ to identify the *recurrent* requests, which are considered to be almost stable from one day to another. This parameter, β, plays a key role in deducing the *reference* request set $\bar{\mathcal{R}}$ from $\mathcal{R}$. The *recurrent* requests should generate very similar reference requests, whereas the remaining requests will be generated to meet the statistical distribution of the requests.

Summary Table 1 summarizes the basic settings of the *RSMoD* system and the requests.

Table 1: Basic setting of the system and the requests

Param	Description	Value
$\lvert\mathcal{N}\rvert$	the number of pick-up and delivery locations	1,496
T	the planning horizon	24 h
I_E	the duration of each decision epoch of each decision epoch	10 min.
V	SAV fleet size	1,977
$\lvert\mathcal{R}\rvert$	the number of dynamic requests	300k
δ^r	the length of pick-up time window	15 min.
T^r	the maximum ride time of any request r	$2t(o^r, d^r)$
q^r	the load of request r	1
$\lvert\bar{\mathcal{R}}\rvert$	the number of reference requests	400k
β	the proportion of "random" part in $\mathcal{R}$	10%

6.2 Fleet sizing and the reference *S_DARP* problem

The *RSMoD* system studied in this paper aims to cater to around 300,000 dynamic requests daily. As explained in Section 5, we compute pertinent fleet sizes by proceeding as follows.

First, we generate six *static* instances of varying sizes that meet the statistical features previously described. These instance sizes range from 300,000 to 425,000, increasing in increments of 25,000. For any static instance, we address the associated static problem *S_DARP* with BF (see Section 4.2) and get the resulting fleet sizes. Table 2 shows the fleet sizes obtained from resolving each of the six static problems.

Next, we generate five trial *dynamic* instances $\{i0, i1, \ldots, i4\}$, with 300,000 dynamic requests. For each instance, we address the six related problems D_DARP while relying on the basic algorithm BF and fixing the fleet size at the above-established six values. We get every time a request rejection rate. Table 3 presents the passenger rejection rates under different fleet sizes for different instances and the corresponding averaged values.

Then we consider 1,977 as the most reasonable fleet size: The passenger rejection rate is close to 0, under significantly smaller economic costs than the other tested size of 2,069. Of course, we might try fleet sizes between 1,977 and 2,069: Here we only propose a tentative approach for the fleet sizing issue.

An advantage of the above process is that it provides us with both the reference request set and the reference route sets required by *GIM*. The static request set of size 400,000 yielding the 1,977 SAVs is selected as the reference set $\bar{\mathcal{R}}$, and the related solution becomes the reference route set $\bar{\Theta}$.

6.3 The behavior of the RSMoD system

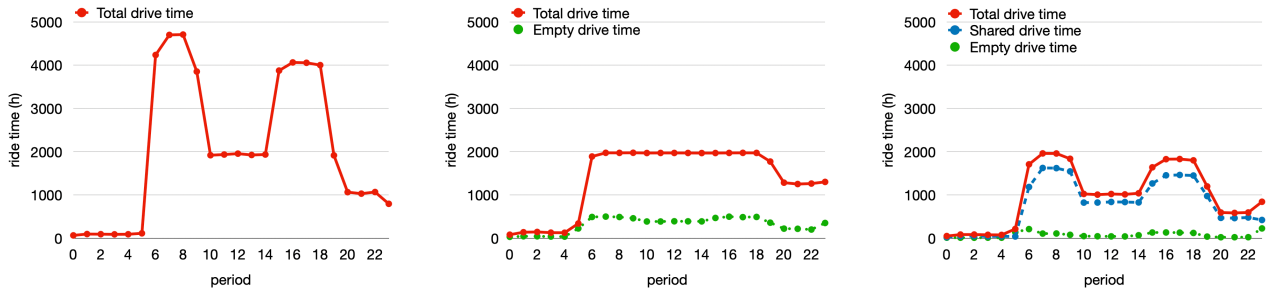
We compare here the performance of the *RSMoD* system to two other systems: *private* and *MoD*. In the *private* system, the 300,000 passenger requests are all supported by private vehicles. We estimate a fleet size of $240,951$ in *private*, which is the number of one-way requests plus the number of symmetric requests. In *MoD*, ride-sharing is not allowed. Its fleet size is 1,977, the same as in *RSMoD*. The baseline approach BF is used to solve the corresponding routing problems.

As shown in Table 4, in *private* system, the total drive time is much higher than *MoD* and *RSMoD*, because much more vehicles are utilized. Almost half of the requests are rejected in *MoD*, but its total drive time is still higher than that in *RSMoD*. Because no ride-sharing is allowed in *MoD*, and SAVs must make the re-positioning to reach different pick-up locations. We see that empty drive time contributes to 31.5% of the total drive time (the column **Relative empty drive time**) in *MoD*. Additionally, according to the results, to cater for all the rejected requests in MoD, an estimated minimum number of additional SAVs of around $2,000$ would be required, resulting in a great increase in the total drive time as well. Remind that most of the rejected requests are associated with peak hours, we would need many more SAVs than $2,000$. In contrast, thanks to ride-sharing, 78.1% of the total drive time is shared by more than one passenger in *RSMoD*, and only 7.4% is issued from empty relocation, with a smaller total drive time compared to the two other system configurations. These results prove several advantages of the *RSMoD* system. First, the on-route vehicles in the city can be largely reduced, thus the traffic congestion and occupancy of the city infrastructure can be improved. And SAVs are self-driving, reducing the cost of human labor. Furthermore, the fact that *RSMoD* decreases the total drive time in our simulation proves that the consumption of energy can be greatly economized by deploying such a novel system. Finally, the passengers' travel comfort remains the same in *MoD*, and is decreased in *RSMoD*, with an average of 16.63 minutes per passenger, suggesting that SAVs make detours to promote the ride-sharing so that the rejection of requests can be minimized.

Figure 5 provides a better vision of how drive times of each category (total drive time, empty drive time, and

Table 2: Fleet sizes to different static problem sizes

Instance size	300,000	325,000	350,000	375,000	400,000	425,000
Fleet size	1,621	1,667	1,768	1,869	1,977	2,069



(a) private (b) MoD (c) RSMoD

Figure 5: Accumulated drive time, empty drive time, and shared drive time (where appropriated) in each period of duration of one hour

Table 3: The rejection rate when solving static problems with different fleet sizes

Fleet size	**Instance**					**Avg**
	i0	**i1**	**i2**	**i3**	**i4**	
1,621	6.2%	6.4%	6.3%	6.2%	6.3%	6.3%
1,667	4.7%	4.8%	4.9%	4.7%	5.0%	4.8%
1,768	2.2%	2.2%	2.1%	2.3%	2.3%	2.2%
1,869	1.0%	0.8%	1.0%	1.0%	1.0%	1.0%
1,977	0.1%	0.0%	0.1%	0.1%	0.1%	0.1%
2,069	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

shared drive time) are distributed across the service horizon. In both *private* and *RSMoD* systems, the drive time during peak hours is much higher than in other time slots. In *MoD*, the total drive time remains almost the same during **MP**, **EP** and **NH**, because the number of requests that can be serviced during these time slots is almost the same (excessive requests during peak hours are rejected). Furthermore, as shown in Sub-figure 5b and Sub-figure 5c, empty drive time in *MoD* is higher than in *RSMoD*, especially during peak hours, because more detours are required to pick up new passengers. And a large proportion of trajectory during **MP**, **EP** and **NH** are shared in *RSMoD*.

6.4 The behavior of the GIM-FS algorithmic scheme

In addition to the two approaches, GIM-PFS and GIM-CFS, we try 4 variants: BF, PFS, CFS, and GIM-BF, of the GIM-FS algorithmic scheme. Whenever FS is utilized, it is configured as follows: the spatial-temporal partition results in 24 periods of equal duration and 20 zones of equal area. In CFS, the stopping criterion $\mathcal{T}^r$ set for each request r is equivalent to exploring at most 10% of the already-activated vehicles. In addition, a *regeneration mechanism* is employed, which proportionally expands the search space when no feasible insertion is present in the first candidate search space. These variants are summarized in Table 6. Furthermore, in GIM, the parameter α that defines the spatial neighborhood of pattern points is fixed at 10 minutes.

6.4.1 General results

Table 7 presents the general results when solving the dynamic problem with different approaches and different values of the request parameter β.

The column **CPU time** represents the accumulated execution time to solve *D_DARP*. Generally speaking, for every β, the two-phase approaches integrated with GIM are more efficient than their basic algorithm, thanks to the largely reduced search space provided by GIM. Meanwhile, when utilizing two-phase approaches, the quality of the solution remains almost the same, compared to those when applying the corresponding basic algorithms. Typically, GIM-BF and GIM-PFS reject slightly more requests and demonstrate a slightly longer total drive time than BF and GIM-BF, respectively. In addition, for every β, GIM-CFS outperforms CFS, in both execution time and solution quality, thanks to the guide provided by GIM in Phase 1. Specifically, although CFS can solve the problem in a very short time thanks to a largely reduced search space, the resulting solution is less satisfactory. Because this reduction is only deduced from the current status of the system in a rather myopic and greedy way, without any consideration of the global quality of the solution. However, GIM utilizes the additional information of the reference resolution and "forces" vehicles to stick to their pre-defined travel patterns extracted from optimal reference routes. This helps

Table 4: Performance of three systems

	Fleet size	Rejection rate	Total drive time (h)	Relative[1] shared drive time	Relative[2] empty drive time	Average in-vehicle time (min)
private	240,951	/	49,585	/	/	9.92
MoD	1,977	48.0%	33,343	/	31.5%	10.30
RSMoD	1,977	0.5%	24,035	78.1%	7.4%	16.63

[1] Relative shared drive time is calculated as the percentage of total drive time that is shared with other passengers.
[2] Relative empty drive time is calculated as the percentage of total drive time where the vehicle is empty.

Table 5: Sensitivity analyses of GIM

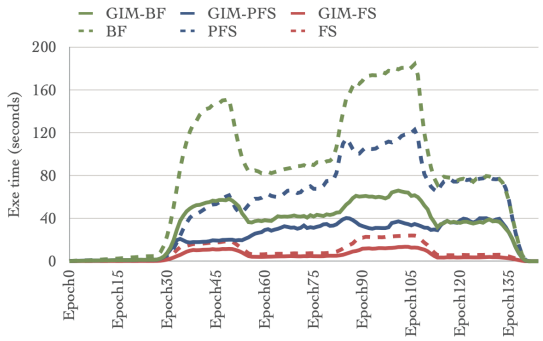
δ^s (min)	δ^t (min)	Approach	CPU time (min)	Rejection rate	Total drive time (h)	Average in-vehicle time (min)	matchGI rate	succGI rate
2	3	GIM-BF	137.1	0.7%	**23,571**	16.39	67.5%	41.0%
		GIM-PFS	93.5	**0.7%**	23,585	**16.38**		41.0%
		GIM-FS	17.9	2.3%	26,725	17.16		36.3%
	7.5	GIM-BF	134.2	0.7%	23,604	16.38	89.6%	40.0%
		GIM-PFS	92.1	0.7%	23,600	**16.37**		40.0%
		GIM-FS	17.6	2.3%	26,713	17.15		34.9%
	15	GIM-BF	85.3	0.9%	24,092	16.45	92.3%	62.0%
		GIM-PFS	56.0	0.9%	24,095	16.45		61.8%
		GIM-FS	13.3	1.7%	26,126	16.87		58.4%
3	3	GIM-BF	82.1	0.8%	24,129	16.40	82.7%	60.4%
		GIM-PFS	54.1	0.8%	24,109	16.41		60.3%
		GIM-FS	12.8	1.6%	26,117	16.8		56.6%
	7.5	GIM-BF	76.5	1.2%	24,497	16.58	95.4%	65.1%
		GIM-PFS	47.9	1.2%	24,498	16.57		65.2%
		GIM-FS	12.6	2.0%	26,190	16.99		62.4%
	15	GIM-BF	71.8	1.2%	24,527	16.53	97.1%	62.6%
		GIM-PFS	45.9	1.2%	24,527	16.5		62.6%
		GIM-FS	11.8	1.8%	26,166	16.87		59.3%
5	3	GIM-BF	68.5	3.1%	25,655	17.51	94.3%	64.2%
		GIM-PFS	40.3	3.2%	25,657	17.55		64.0%
		GIM-FS	12.5	4.0%	27,048	17.92		62.3%
	7.5	GIM-BF	44.8	2.7%	25,817	17.36	99.0%	76.6%
		GIM-PFS	26.3	2.7%	25,809	17.37		76.5%
		GIM-FS	**11.6**	3.0%	26,671	17.52		76.0%
	15	GIM-BF	37.3	2.8%	26,035	17.4	**99.5%**	**81.0%**
		GIM-PFS	22.7	2.8%	26,033	17.42		80.7%
		GIM-FS	12.4	3.0%	26,668	17.49		80.6%

The best results among all combinations are presented in bold font.

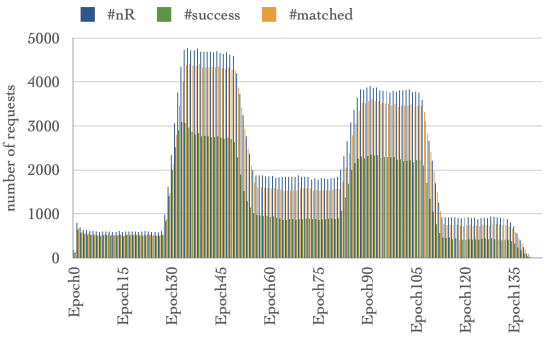
have a sort of pre-concern of future requests in the current insertion, thus making up for the inefficiency of CFS.

The reduction in CPU time by GIM-BF and GIM-PFS compared to BF and PFS, respectively, becomes stronger when there are more similar requests between the real request set and the reference request set. When more requests have similar reference requests, the proportion of requests that can be inserted via GIM should be higher, making the execution faster. However, this reduction effect between GIM-CFS and CFS across different scenarios is less prominent, for the already short enough execution time.

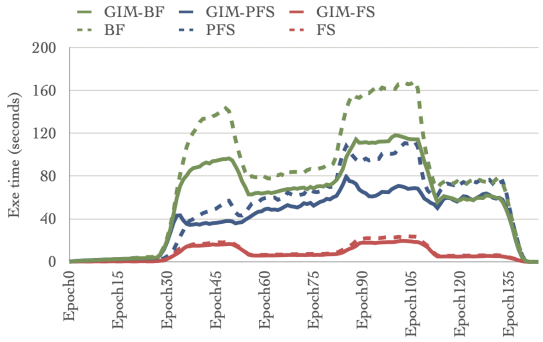
A closer analysis of the efficiency of the approaches can



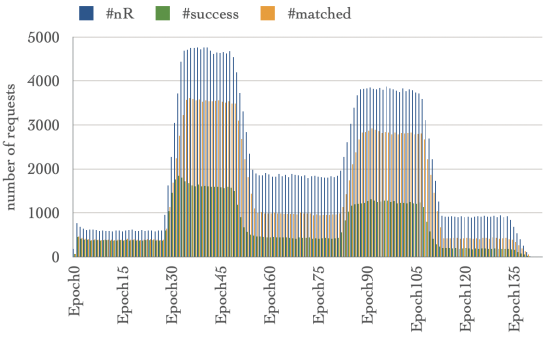
(a) CPU time per epoch ($\beta = 0.9$)



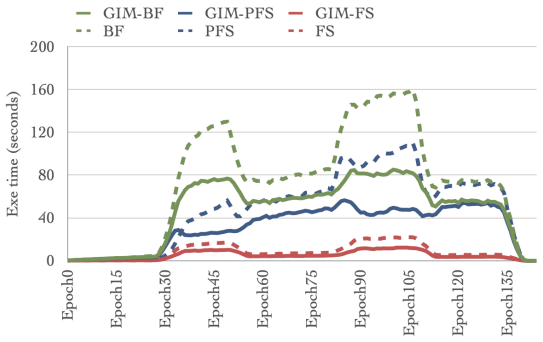
(b) GIM effectiveness ($\beta = 0.9$)



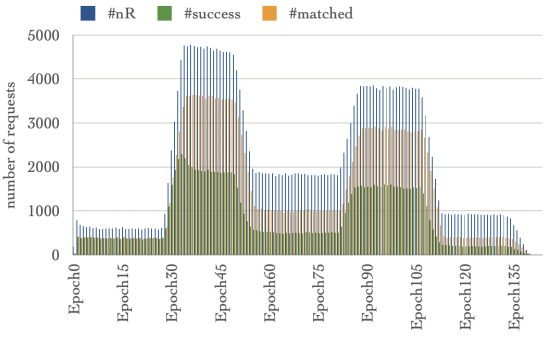
(c) CPU time per epoch ($\beta = 0.5$)



(d) GIM effectiveness ($\beta = 0.5$)



(e) CPU time per epoch ($\beta = 0.3$)



(f) GIM effectiveness ($\beta = 0.3$)

Figure 6: Effectiveness of the algorithm. Each row corresponds to a value β. Sub-figures in the first column represent the CPU time spent to solve the sub routing problem for each decision epoch when using different approaches. Sub-figures in the second column represent the number of requests to be processed (blue), that are associated with at least one similar reference request in GIM (orange), and that are successfully inserted via GIM (green)

be done through Figure 6, which shows different statistics in every decision epoch. In terms of the CPU time spent on the insertion of requests in each decision epoch, we observe that approaches integrated with GIM spend much less time regarding their basic approach. For the purpose of thorough analyses, we did not set a specific value of η. From Sub-figures 6a, 6c and 6e, we see that when setting η at different values, several methods may fail to complete during certain epochs: When $\eta = 40$ seconds, only CFS and GIM-CFS can finish all insertions in every epoch.

The efficiency of two-phase approaches can be explained by the number of successful insertions via GIM. For example, from Sub-figures 6b, 6d and 6f, we note that with $\beta = 0.9$, the number of requests that have at least one similar reference request (orange bars in figures), and the successful insertions via GIM (green bars in figures) are higher than with the other two β values, especially during peak hours. Therefore, fewer requests are passed to Phase 2 when $\beta = 0.9$, and the insertion process becomes faster.

Table 7: General test results with different approaches under different β

(a) $\beta = 0.9$

Approach	CPU time (min)	#rejection	Rejection rate	Total drive time (h)	Average in-vehicle time (min)
BF	200.4	1,631	0.5%	24,035	16.63
PFS	130.6	1,624	0.5%	24,040	16.63
CFS	21.8	10,320	3.4%	27,486	15.99
GIM-BF	82.1	2,420	0.8%	24,129	16.40
GIM-PFS	54.1	2,493	0.8%	24,109	16.41
GIM-CFS	12.8	4,682	1.6%	26,117	16.80

(b) $\beta = 0.5$

Approach	CPU time (min)	#rejection	Rejection rate	Total drive time (h)	Average in-vehicle time (min)
BF	188.6	2,725	0.9%	24,306	16.67
PFS	123.1	2,674	0.9%	24,274	16.68
CFS	21.5	11,969	4.0%	27,594	16.02
GIM-BF	108.6	3,792	1.3%	24,189	16.71
GIM-PFS	72.6	3,762	1.3%	24,190	16.69
GIM-CFS	15.2	8,136	2.7%	26,746	17.43

(c) $\beta = 0.3$

Approach	CPU time (min)	#rejection	Rejection rate	Total drive time (h)	Average in-vehicle time (min)
BF	178.1	2,093	0.7%	24,131	16.73
PFS	118.0	1,973	0.7%	24,104	16.73
CFS	20.2	10,798	3.6%	27,512	16.10
GIM-BF	115.7	3,270	1.1%	23,978	16.69
GIM-PFS	76.0	3,302	1.1%	23,952	16.69
GIM-CFS	12.3	8,128	2.7%	26,839	17.49

Table 6: Description of tested approaches

Approach	**Description**
BF	Best-Fit insertion heuristic
PFS	Partial Filtering System
CFS	Complete Filtering System
GIM-BF	GIM is used in Phase 1, BF is used in Phase 2
GIM-PFS	GIM is used in Phase 1, PFS is used in Phase 2
GIM-CFS	GIM is used in Phase 1, CFS is used in Phase 2

6.4.2 Sensitivity analysis

From the above discussion, we see that the rate of successful insertions via GIM has a great impact on the efficiency of the two-phase methods. This rate is directly influenced by the number of requests that can be matched to at least one similar reference request. Two parameters, δ^s and δ^t, control the similarity measure, thus the match rate. From now on, we only focus on the $\beta = 0.9$, and analyze the sensitivity of the two-phase algorithms while varying the values of δ^s and δ^t.

Table 5 shows the performances of different methods under different δ^s and δ^t combinations.

Typically, the column *matchGI rate* represents the percentage of requests that have at least one similar reference request, and the column *succGI rate* computes the percentage of requests that are successfully inserted via GIM. We conclude that when the similarity measure becomes more relaxed (bigger δ^s and δ^t), the matchGI rate and succGI rate effectively become bigger. For example, in the most relaxed situation, 99.5% of requests are passed to GIM, and 81% of requests are successfully inserted from GIM. We confirm that the general trend is that a higher succGI rate implies a smaller CPU time. However, a bigger succGI rate is not equivalent to a better solution. We get the opposite situation instead. Because such an increase in succGI rate is gained from a more relaxed request similarity measure. Consequently, as long as a feasible insertion is found in Phase 1, GIM may force the request to be inserted by mimicking the insertion patterns of a not-very-similar reference request. In the short term, the insertion of such a request is based on a relatively bad candidate insertion parameter. In the long term, such insertion may lead to a distortion of the trajectory of the vehicle in question from its pre-defined pattern, preventing later perfectly matched requests from being inserted, thus worsening the quality of the solution.

7 Conclusion

In this paper, we studied a prospective large-scale *RSMoD* transportation system with SAVs. We showed that this SAV *RSMoD* system would provide on-demand services with a very low rejection rate and satisfactory passenger riding comfort, while, thanks to its ride-sharing feature, significantly reducing the total driving time, energy costs, and the number of on-route vehicles.

As for the operational management of such a system, we introduced a GIM: *Guided Insertion Mechanism*, that learns from a reference solution deriving from a statistical virtual version of the related DARP, and combined it with a Filtering System to form a two-phase algorithm framework, GIM-FS. We proved that GIM is likely to significantly help the manager of the system efficiently assign dynamic requests to the vehicles.

Yet, several issues remain that will motivate our future research. First, we must try (algorithmic issue) to improve the basic greedy insertion heuristic used to solve the statistical virtual version of the related DARP, in a way that will fit the very large-scale feature and that will allow us to make the GIM component more efficient. Next, we should integrate into our decision-making framework the energetic issue, and thus take into account that the schedule of a vehicle should include its recharge transactions and the time-varying costs of these recharge transactions. Finally, we should address the robustness issue, related to the non-deterministic features of any mass *RSMoD* system: traffic congestion, passenger no-shows, etc.

Acknowledgement

This work was supported by the International Research Center “Innovation Transportation and Production Systems” of the I-SITE CAP 20-25.

References

[1] Ransford A. Acheampong, Alhassan Siiba, Dennis K. Okyere, and Justice P. Tuffour. Mobility-on-demand: An empirical study of internet-based ride-hailing adoption factors, travel characteristics and mode substitution effects. Transportation Research Part C: Emerging Technologies, 115:102638, June 2020. https://doi.org/10.1016/j.trc.2020.102638.

[2] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. European Journal of Operational Research, 223(2):295–303, December 2012. https://doi.org/10.1016/j.ejor.2012.05.028.

[3] Vincent Armant and Kenneth N. Brown. Fast optimised ridesharing: Objectives, reformulations and driver flexibility. Expert Systems with Applications, 141:112914, March 2020. https://doi.org/10.1016/j.eswa.2019.112914.

[4] Bilge Atasoy, Takuro Ikeda, Xiang Song, and Moshe E. Ben-Akiva. The concept and impact analysis of a flexible mobility on demand system. Transportation Research Part C: Emerging Technologies, 56:373–392, July 2015. https://doi.org/10.1016/j.trc.2015.04.009.

[5] Slim Belhaiza. A Hybrid Adaptive Large Neighborhood Heuristic for a Real-Life Dial-a-Ride Problem. Algorithms, 12(2):39, February 2019. https://doi.org/10.3390/a12020039.

[6] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. A Hybrid Tabu Search and Constraint Programming Algorithm for the Dynamic Dial-a-Ride Problem. INFORMS Journal on Computing, May 2011. https://doi.org/10.1287/ijoc.1110.0454.

[7] Kris Braekers, An Caris, and Gerrit K. Janssens. Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. Transportation Research Part B: Methodological, 67:166–186, September 2014. https://doi.org/10.1016/j.trb.2014.05.007.

[8] Roberto Wolfler Calvo and Alberto Colorni. An effective and fast heuristic for the Dial-a-Ride problem. 4OR, 5(1):61–73, March 2007. https://doi.org/10.1007/s10288-006-0018-0.

[9] CEU. MOVE., Univ. Eiffel., TRT., Panteia., GDCC., and STRATEC. Study on new mobility patterns in European cities: final report. Task A, EU wide passenger mobility survey. Publications Office, LU, 2022. https://data.europa.eu/doi/10.2832/728583.

[10] Jean-François Cordeau. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. Operations Research, 54(3):573–586, 2006. https://doi.org/10.1287/opre.1060.0283.

[11] Jean-François Cordeau and Gilbert Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B: Methodological, 37(6):579–594, July 2003. https://doi.org/10.1016/S0191-2615(02)00045-0.

[12] Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. Annals of Operations Research, 153(1):29–46, June 2007. https://doi.org/10.1007/s10479-007-0170-8.

[13] Jean-François Cordeau, Gilbert Laporte, Jean-Yves Potvin, and Martin W. P. Savelsbergh. Chapter 7 Transportation on Demand. In Cynthia Barnhart and Gilbert Laporte, editors, Handbooks in Operations Research and Management Science, volume 14 of Transportation, pages 429–466. Elsevier, January 2007. https://doi.org/10.1016/S0927-0507(06)14007-4.

[14] Samuel Deleplanque and Alain Quilliot. Insertion techniques and constraint propagation for the DARP. In 2012 Federated Conference on Computer Science and Information Systems (FedCSIS), pages 393–400, September 2012.

[15] Daniel J. Fagnant and Kara M. Kockelman. Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas. Transportation, 45(1):143–158, January 2018. https://doi.org/10.1007/s11116-016-9729-z.

[16] Jeffery B. Greenblatt and Susan Shaheen. Automated Vehicles, On-Demand Mobility, and Environmental Impacts. Current Sustainable/Renewable Energy Reports, 2(3):74–81, September 2015. https://doi.org/10.1007/s40518-015-0038-5.

[17] Martin L Hazelton. Inference for origin–destination matrices: estimation, prediction and reconstruction. Transportation Research Part B: Methodological, 35(7):667–676, August 2001. https://doi.org/10.1016/S0191-2615(00)00009-6.

[18] Sin C. Ho, W.Y. Szeto, Yong-Hong Kuo, Janny M.Y. Leung, Matthew Petering, and Terence W.H. Tou. A survey of dial-a-ride problems: Literature review and recent developments. Transportation Research Part B: Methodological, 111:395–421, May 2018. https://doi.org/10.1016/j.trb.2018.02.001.

[19] Zihan Hong, Ying Chen, Hani S. Mahmassani, and Shuang Xu. Commuter ride-sharing using topology-based vehicle trajectory clustering: Methodology, application and impact evaluation. Transportation Research Part C: Emerging Technologies, 85:573–590, December 2017. https://doi.org/10.1016/j.trc.2017.10.020.

[20] Esa Hyytiä, Lauri Häme, Aleksi Penttinen, and Reijo Sulonen. Simulation of a large scale dynamic pickup and delivery problem. In Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, Malaga, Spain, 2010. ICST. https://doi.org/10.4108/ICST.SIMUTOOLS2010.8701.

[21] Siddhartha Jain and Pascal Van Hentenryck. Large Neighborhood Search for Dial-a-Ride Problems. In Jimmy Lee, editor, Principles and Practice of Constraint Programming – CP 2011, volume 6876, pages 400–413. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. https://doi.org/10.1007/978-3-642-23786-7_31.

[22] Jang-Jei Jaw, Amedeo R. Odoni, Harilaos N. Psaraftis, and Nigel H. M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. Transportation Research Part B: Methodological, 20(3):243–257, June 1986. https://doi.org/10.1016/0191-2615(86)90020-2.

[23] Jiwon Kim and Hani S. Mahmassani. Spatial and Temporal Characterization of Travel Patterns in a Traffic Network Using Vehicle Trajectories. Transportation Research Procedia, 9:164–184, 2015. https://doi.org/10.1016/j.trpro.2015.07.010.

[24] Jiwon Kim, Kai Zheng, Jonathan Corcoran, Sanghyung Ahn, and Marty Papamanolis. Trajectory Flow Map: Graph-based Approach to Analysing Temporal Evolution of Aggregated Traffic Flows in Large-scale Urban Networks, December 2022. https://doi.org/10.48550/arXiv.2212.02927.

[25] Da Lei, Xuewu Chen, Long Cheng, Lin Zhang, Satish V. Ukkusuri, and Frank Witlox. Inferring temporal motifs for travel pattern analysis using large scale smart card data. Transportation Research Part C: Emerging Technologies, 120:102810, November 2020. https://doi.org/10.1016/j.trc.2020.102810.

[26] Michael W. Levin. Congestion-aware system optimal route choice for shared autonomous vehicles. Transportation Research Part C: Emerging Technologies, 82:229–247, September 2017. https://doi.org/10.1016/j.trc.2017.06.020.

[27] Chijia Liu, Alain Quilliot, Hélène Toussaint, and Dominique Feillet. A filtering system to solve the large-scale shared autonomous vehicles Dial-a-Ride Problem. Transportation Research Part C: Emerging Technologies, 161:104551, April 2024. https://doi.org/10.1016/j.trc.2024.104551.

[28] Benjamin Loeb, Kara M. Kockelman, and Jun Liu. Shared autonomous electric vehicle (SAEV) operations across the Austin, Texas network with charging infrastructure decisions. Transportation Research Part C: Emerging Technologies, 89:222–233, April 2018. https://doi.org/10.1016/j.trc.2018.01.019.

[29] J. Carlos Martínez Mori, M. Grazia Speranza, and Samitha Samaranayake. On the Value of Dynamism in Transit Networks. Transportation Science, 57(3):578–593, May 2023. https://doi.org/10.1287/trsc.2022.1193.

[30] Santhanakrishnan Narayanan, Emmanouil Chaniotakis, and Constantinos Antoniou. Shared autonomous vehicle services: A comprehensive review. Transportation Research Part C: Emerging Technologies, 111:255–293, February 2020. https://doi.org/10.1016/j.trc.2019.12.008.

[31] Julie Paquette, Jean-François Cordeau, Gilbert Laporte, and Marta M. B. Pascoal. Combining multicriteria analysis and tabu search for dial-a-ride problems. Transportation Research Part B: Methodological, 52:1–16, June 2013. https://doi.org/10.1016/j.trb.2013.02.007.

[32] Sophie N. Parragh, Jorge Pinho de Sousa, and Bernardo Almada-Lobo. The Dial-a-Ride Problem with Split Requests and Profits. Transportation Science, 49(2):311–334, 2015. https://doi.org/10.1287/trsc.2014.0520.

[33] Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl. Variable neighborhood search for the dial-a-ride problem. Computers & Operations Research, 37(6):1129–1138, June 2010. https://doi.org/10.1016/j.cor.2009.10.003.

[34] Sophie N. Parragh and Verena Schmid. Hybrid column generation and large neighborhood search for the dial-a-ride problem. Computers & Operations Research, 40(1):490–497, January 2013. https://doi.org/10.1016/j.cor.2012.08.004.

[35] Ronik Ketankumar Patel, Roya Etminani-Ghasrodashti, Sharareh Kermanshachi, Jay Michael Rosenberger, and Ann Foss. Mobility-on-demand (MOD) Projects: A study of the best practices adopted in United States. Transportation Research Interdisciplinary Perspectives, 14:100601, June 2022. https://doi.org/10.1016/j.trip.2022.100601.

[36] Ronik Ketankumar Patel, Roya Etminani-Ghasrodashti, Sharareh Kermanshachi, Jay Michael Rosenberger, and Ann Foss. Exploring willingness to use shared autonomous vehicles. International Journal of Transportation Science and Technology, 12(3):765–778, September 2023. https://doi.org/10.1016/j.ijtst.2022.06.008.

[37] Marco Pavone. Autonomous Mobility-on-Demand Systems for Future Urban Mobility. In Markus Maurer, J. Christian Gerdes, Barbara Lenz, and Hermann Winner, editors, Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte, pages 399–416. Springer, Berlin, Heidelberg, 2015. https://doi.org/10.1007/978-3-662-45854-9_19.

[38] Harilaos N. Psaraftis. A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem. Transportation Science, 14(2):130–154, May 1980. https://doi.org/10.1287/trsc.14.2.130.

[39] Harilaos N. Psaraftis. An Exact Algorithm for the Single Vehicle Many-to-Many Dial-A-Ride Problem with Time Windows. Transportation Science, 17(3):351–357, August 1983. https://doi.org/10.1287/trsc.17.3.351.

[40] Lisa Rayle, Danielle Dai, Nelson Chan, Robert Cervero, and Susan Shaheen. Just a better taxi? A survey-based comparison of taxis, transit, and ridesourcing services in San Francisco. Transport Policy, 45:168–178, January 2016. https://doi.org/10.1016/j.tranpol.2015.10.004.

[41] M. Schilde, K. F. Doerner, and R. F. Hartl. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. Computers & Operations Research, 38(12):1719–1730, December 2011. https://doi.org/10.1016/j.cor.2011.02.006.

[42] Susan Shaheen and Adam Cohen. Shared ride services in North America: definitions, impacts, and the future of pooling. Transport Reviews, 39(4):427–442, July 2019. https://doi.org/10.1080/01441647.2018.1497728.

[43] Shuo Ma, Yu Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In 2013 IEEE 29th International Conference on Data Engineering (ICDE), pages 410–421, Brisbane, QLD, April 2013. IEEE. https://doi.org/10.1109/ICDE.2013.6544843.

[44] Sacha Varone and Vytenis Janilionis. Insertion heuristic for a dynamic dial-a-ride problem using geographical maps. In MOSIM 2014, 10ème Conférence Francophone de Modélisation, Optimisation et Simulation, Nancy, France, November 2014. https://hal.science/hal-01166662.

[45] Yuandong Wang, Hongzhi Yin, Hongxu Chen, Tianyu Wo, Jie Xu, and Kai Zheng. Origin-Destination Matrix Prediction via Graph Convolution: a New Perspective of Passenger Demand Modeling. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1227–1235, Anchorage AK USA, July 2019. ACM. https://doi.org/10.1145/3292500.3330877.

[46] Salomon Wollenstein-Betech, Mauro Salazar, Arian Houshmand, Marco Pavone, Ioannis Ch. Paschalidis, and Christos G. Cassandras. Routing and Rebalancing Intermodal Autonomous Mobility-on-Demand Systems in Mixed Traffic. IEEE Transactions on Intelligent Transportation Systems, 23(8):12263–12275, August 2022. https://doi.org/10.1109/TITS.2021.3112106.

[47] Zixuan Yu, Ping Zhang, Yang Yu, Wei Sun, and Min Huang. An Adaptive Large Neighborhood Search for the Larger-Scale Instances of Green Vehicle Routing Problem with Time Windows. Complexity, 2020:1–14, October 2020. https://doi.org/10.1155/2020/8210630.

[48] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. Proceedings of the AAAI Conference on Artificial Intelligence, 31(1), February 2017. https://doi.org/10.1609/aaai.v31i1.10735.

[49] Linjiang Zheng, Dong Xia, Xin Zhao, Longyou Tan, Hang Li, Li Chen, and Weining Liu. Spatial–temporal travel pattern mining using massive taxi trajectory data. Physica A: Statistical Mechanics and its

Applications, 501:24–41, July 2018. https://doi.org/10.1016/j.physa.2018.02.064.