A Guided Insertion Mechanism for Solving the Dynamic Large-Scale Dial-a-Ride Problem

Chijia Liu
chijia.liu@uca.fr
Université Clermont Auvergne, CNRS, Clermont Auvergne INP, Mines Saint-Etienne, UMR 6158 LIMOS
Clermont-Ferrand, France

Quilliot Alain
quilliot.alain@uca.fr
Université Clermont Auvergne, CNRS, Clermont Auvergne INP, Mines Saint-Etienne, UMR 6158 LIMOS
Clermont-Ferrand, France

Hélène Toussaint
helene.toussaint@uca.fr
Université Clermont Auvergne, CNRS, Clermont Auvergne INP, Mines Saint-Etienne, UMR 6158 LIMOS
Clermont-Ferrand, France

Dominique Feillet
feillet@emse.fr
Mines Saint-Etienne, Univ Clermont Auvergne, Clermont Auvergne INP, CNRS, UMR 6158 LIMOS, F
Saint-Etienne, France

ABSTRACT

We study a prospective transportation system where vehicles provide dial-a-ride services to fulfill a very large scale of passenger requests (around 300,000). The system operates dynamically, with newly submitted requests needing immediate processing. A crucial aspect of this system's viability in real-time situations is therefore the implementation of an efficient routing algorithm that can deliver high-quality solutions. We address a dynamic large-scale dial-a-ride problem through a best-fit greedy insertion algorithm. Furthermore, large-scale requests are assumed to be dominated by daily commuting needs and thus should be similar and repetitive from one day to another. Therefore, there might exist recurring and similar patterns in vehicle trajectories if similar requests can be served in the same manner. We introduce a Guided Insertion Mechanism that relies on a representative reference resolution and guides the insertion of dynamic requests while maintaining high-quality solutions.

1 INTRODUCTION

In recent years, we have witnessed the emergence of novel transportation systems, offering efficient and sustainable alternatives to traditional modes of transportation. Among these, *vehicle-sharing* and *ride-sharing* systems are two predominant categories. In vehicle-sharing systems, vehicles positioned at stations are left for access by users. Related problems can be about the design of the systems, such as where to position the stations and how to relocalize vehicles [8]. Ride-sharing services allow users to share common journeys in the same vehicle. The most common problems are about the effective routing and scheduling of vehicles that take advantage of the mutualization of different services [1, 7]. Both systems promote reduced congestion and emissions while maximizing vehicular utilization. *Dail-A-Ride* (DAR) system can be regarded as a hybrid of the above two categories, where vehicles are owned by operators and provide demand-responsive transportation services to fulfill the diverse needs of various communities and enhance accessibility across urban transit systems.

In this paper, we consider a prospective transportation system where a mid-capacity vehicle fleet offers Dial-A-Ride (DAR) services to a very large volume of passengers (around 300, 000 per day), catering to the transportation needs of a vast user base. The system operates in a dynamic context, where user requests are submitted and processed on-the-fly. Therefore, to ensure the viability of the system, we need to implement a routing algorithm that is both efficient and capable of delivering high-quality solutions. For that, we address a dynamic large-scale dial-a-ride problem. Due to the large-scale aspect, it is difficult to solve the problem through exact methods(e.g., branch and bound, branch and cut, etc.) or local search approaches (e.g., Large Neighborhood Search, Build and Destroy, etc). We rely on the classic greedy insertion heuristic for its simplicity and effectiveness. Furthermore, we consider that large-scale requests should be dominated by daily commute needs which exhibit recurring characteristics. Therefore, daily requests should be globally similar and repetitive. Under this assumption, we should be able to recreate similar travel patterns in vehicle trajectories. Based on this idea, as our main contribution to this work, we propose a *Guided Insertion Mechanism (GIM)* which utilizes the travel patterns constructed from a representative reference resolution to efficiently insert dynamic requests while maintaining the high quality of the routing solution.

The remainder of this paper is organized as follows: In Section 2 we introduce some related works in the literature. Section 3 formally defines our problem and introduces some important notation. Section 4 introduces the notion of insertion of a request. In Section 5, the solution scheme is outlined. Then, in Section 6, we detail the *GIM*. Experimental results are presented and discussed in Section 7 and we conclude in Section 8.

2 RELATED WORKS

On-demand DAR transportation systems provide personalized and efficient transportation services to customers. These systems utilize digital platforms and shared mobility concepts, offering convenient, flexible, and cost-effective travel choices. As the need for efficient urban transportation increases, there is a rising prevalence of studies focusing on large-scale systems. Studies are actively studying

various system characteristics, encompassing the management of congestion issues resulting from the substantial vehicle fleet size [13], addressing recharging challenges in scenarios involving electric vehicles [2], and exploring the potential advantages of introducing ride-sharing into these systems [6]. In customer-facing systems like these, the rapid processing of user demands during dynamic scenarios is crucial. Hence, expediting the routing and scheduling process holds vital significance. The authors in [16] study a large-scale taxi system that serves at least 330, 000 requests per day. To provide instant feedback in response to dynamic requests, they introduce a filtering algorithm *dual-side taxi searching* that rapidly retrieves some interesting (nearest) vehicles to be candidates to service the given request. Similarly, in [14], the authors propose a filtering system that not only provides candidate vehicles but corresponding candidate insertion positions within the routes as well.

Within the context of DAR transportation systems, we naturally rely on the Dial-A-Ride Problem (DARP) to formulate the problem. Designing a fleet of vehicles' routes and schedules to accommodate a set of passenger requests—typically defined by a pickup location, a drop-off location, pickup and/or drop-off time windows, and a maximum ride time—is the basis of DARP. Objectives like maximizing customer service quality or reducing vehicle operating costs guide decisions [10]. DARP is NP-hard as it admits the Vehicle Routing Problem (VRP) as a special case. Very few studies seek to solve the DARP using exact optimization methods, unless for solving small-sized static problems [4, 15]. More studies tend to propose approximate methods capable of solving larger instances, such as tabu search meta-heuristics [5], Large Neighborhood Search (LNS) [9]. For example, [3] proposes a two-phase scheduling heuristic that first builds an auxiliary graph and then solves an assignment problem on this graph.

Furthermore, in a very large-scale context, daily travel requests are supposed to obey a certain repetitive pattern because they should be dominated by regular commuting needs. This makes capturing the daily mobility patterns of requests a hot topic in the literature. If sufficient historical daily instances are available, this task can be achieved through some machine learning or deep learning approaches [18, 19]. On the other hand, due to the similarity in passengers' travel requests, the trajectories of vehicles supporting these requests should also exhibit some regular travel patterns. In [12], the authors propose a graph-based analysis framework that characterizes spatial and temporal patterns of network-wide traffic flows. In [11], a trajectory clustering method is presented to discover spatial and temporal travel patterns in a traffic network. In this paper, we do not take care of extracting travel patterns from historical events and assume that a sufficiently representative reference request instance in our DAR system is available. We introduce a *Guided Insertion Mechanism (GIM)* that uses a set of simplified routes that we call *vehicle travel patterns* established by solving the related reference problem to help solve a large-scale dynamic DARP more quickly. To the best of our knowledge, we are the first to consider the correlation between vehicle travel patterns and the resolution of DARP. Additionally, we offer a formal representation of these vehicle travel patterns, providing a framework applicable in resolving DARP.

3 PROBLEM STATEMENT

In this section, we define our Large-Scale Dial-A-Ride Problem (**LSDARP**).

We consider a transit network $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ contains all the intersections, and $\mathcal{A}$ contains all the arcs in the network. We only consider one depot, denoted by $n_0 \in \mathcal{N}$ in $\mathcal{G}$. The travel time of an arc $(i, j) \in \mathcal{A}$, $i, j \in \mathcal{N}^2$, is denoted by $t(i, j)$. By extension, we use $t(u, v)$ to denote the shortest travel time from any node u to any node v in the network.

A request $r \in \mathcal{R}$ is submitted at time t^r_{sub} with the following information: a pickup service requirement O^r, a drop-off service requirement D^r and the number of involved passengers q^r. The pickup service O^r includes an origin $o^r \in \mathcal{N}$ and a pickup time window $[e_{O^r}, l_{O^r}]$. And the drop-off service D^r includes a destination $d^r \in \mathcal{N}$ and a maximum ride time T^r. We note that service times are not considered, and all requests are supposed to be feasible and non-preemptive, which means that each request must be fulfilled exactly once by exactly one vehicle.

Passengers are serviced by a fleet $\mathcal{V}$ of vehicles of capacity Q. A route $\theta^v \in \Theta$ followed by a vehicle v is a list of *key points* K that aggregates services happening at the same location at the same time. Typically, a key point K contains: $n_K \in \mathcal{N}$, the location of the service; q_K, the load of v before departing from n_K; R^+_K, the list of requests scheduled to get onboard at K; R^-_K, the list of requests scheduled to get off at K; $[e^a_K, l^a_K]$, the arrival time window at n_K; and $[e^d_K, l^d_K]$, the departure time window from n_K. Let $succ(K)$ denote the successive key point of K. For any request r assigned to v, we use $K(O^r) \in \theta^v$ (resp. $K(D^r)$) to denote the key point where O^r (resp. D^r) is inserted.

When providing services, v follows the earliest arrival time e^a_K and earliest departure time e^d_K. Vehicle routes must start, end at the depot n_0, and have a load that never exceeds vehicle capacity. And for every request r assigned to the vehicle, O^r precedes D^r, and the schedule must not violate the pickup time window and the maximum ride time constraints.

We consider a lexicographic objective function. We assume that the number of vehicles is unlimited, so minimizing the fleet size is the primary objective. The total drive time of vehicles is considered the second criterion.

4 INSERTION OF A REQUEST

Given a triplet of *insertion parameters* (θ^v, K^o, K^d), we introduce the procedure $INSERTION(r, \theta^v, K^o, K^d)$ that inserts r into the route θ^v at positions K^o and K^d.

We first insert the pickup service O^r. If $n_{O^r} = n_{K^o}$, we *aggregate* O^r to K^o and $K(O^r) = K^o$; otherwise, a new key point $K(O^r)$ supporting O^r will be inserted between K^o and $succ(K^o)$. In both cases, the load $q_{K(O^r)}$, in inbound request list $R^+_{K(O^r)}$, and the pickup and drop-off time windows on $K(O^r)$ should be updated while considering the constraints about the vehicle and requests mentioned previously. Same rules applied for the insertion of the drop-off service D^r at K^d. We note that when updating the arrival time windows of $K(D^r)$, we need to take into account the maximum ride time T^r of r.

Once O^r and D^r are inserted, we increase the load of key points between $K(O^r)$ and $K(D^r)$ by q^r while ensuring that the new loads never exceed the vehicle capacity Q. We should also update the time windows of the key points along θ^v and checking that these time windows are never empty. For that, we implement a classic *constraint propagation procedure* ([17]) considering the above time constraints. The procedure has a complexity of $O(|\theta^v|^2)$, where $O(|\theta^v|^2)$ is the number of key points in θ^v.

5 ALGORITHM FRAMEWORK

We define a set of decision epochs $\mathcal{E} = \{E_0, E_1, \ldots, E_i, \ldots, E_{|E|-1}\}$. Each decision epoch lasts I_e (for example, $I_e = 10$ min) time units. The starting time of E_i is $t_{E_i} = i \times I_e$. For each decision epoch $E_i \in \mathcal{E}$,

(1) during the time slot $[t_{E_i}, t_{E_i} + \tau]$, where τ defines the maximum decision duration, the system makes the routing decisions for requests submitted during the previous epoch E_{i-1}, denoted by $\mathcal{R}_{E_{i-1}}$;
(2) at time $t = t_{E_i} + \tau$, the system updates the vehicles' schedules, and informs unserved passengers whose requests have already been inserted about the updated information about their pickup (the vehicle's passage time);
(3) following the update, vehicles start implementing the new routes until reaching $t = t_{E_{i+1}} + \tau$.

Regarding the very large size of the problem and the need to make decisions on-the-fly, we address the problem to be solved for each decision epoch E based on the classic best-fit insertion heuristic. During epoch E, requests $\mathcal{R}_E$ are inserted one by one following a specific order, and for each request r, we try to insert it according to the best-fitted insertion parameters that minimize the insertion cost measured by the detour to service r. Depending on the implementation, searching for insertion parameters among the whole search space Θ would generally require a computational effort of $O(|\theta^v|^2)$. In addition, testing the insertion feasibility and the *INSERTION* process are also computationally expensive, as mentioned in Section 4. Due to the large-scale effect, the search space would contain thousands of vehicles along with hundreds of thousands of insertion positions to explore, which can be too large to fit the dynamic need. For that, we introduce a *guided insertion mechanism* upstream of the best-fit insertion that rapidly and wisely selects well-fitted insertion parameters and tries several valuable insertions.

Then, given a request $r \in \mathcal{R}$ and the current route collection $\Theta = \{\theta^v, v \in \mathcal{V}\}$:

- We first invoke the *Guided Insertion Mechanism (GIM)* which uses knowledge learned from representative historical instances and solutions to guide the insertion of r. If a feasible insertion is found, we keep it.
- If *GIM* fails, we invoke a best-fit insertion heuristic over the entire search space Θ, and try to insert r into the best-fitted vehicle route at the insertion positions (i.e., key points) that minimize the insertion cost.
- Finally, if both of the above steps fail, we activate a new vehicle v to serve r and add θ^v to the current route set Θ.

6 THE GUIDED INSERTION MECHANISM

In this section, we introduce the *Guided Insertion Mechanism (GIM)*. Let us use **LSDARP**($\mathcal{R}$) to denote the problem with an input instance $\mathcal{R}$. Consider two *similar* request sets $\mathcal{R}_1$ and $\mathcal{R}_2$ in a way that for most of the requests in $\mathcal{R}_1$, we can find a *similar* request in $\mathcal{R}_2$. Then we believe that the optimal routing solution Θ_1 to the problem **LSDARP**($\mathcal{R}_1$) should be similar to Θ_2, the optimal solution to the problem **LSDARP**($\mathcal{R}_2$). Because if $r_1 \in \mathcal{R}_1$ and $r_2 \in \mathcal{R}_2$ are similar, they should be able to be inserted in a similar manner.

GIM is conceived based on the above idea. Thanks to the large-scale aspect and the fact that requests should be dominated by recurring daily commute requests, we assume that requests to be processed in the DAR systems are similar from one day to another. Therefore, the travel patterns of vehicles should also be similar from one day to another. Assuming that we have a representative reference set $\bar{\mathcal{R}}$ which captures the basic distribution (origin and destination and pickup times) of requests, then the *static* (i.e., *off-line*) optimal solution $\bar{\Theta}$ to the problem **LSDARP**($\bar{\mathcal{R}}$) should be able to guide any *dynamic* (i.e., *on-line*) resolution of any real problem **LSDARP**($\mathcal{R}$), where $\mathcal{R}$ is the set of real dynamic requests to be processed in the service period.

Extracting the daily mobility patterns of requests in an underlined system and solving the associated static problem to optimal are two independent problems. As mentioned previously, in this paper, we do not take care of either of the above-mentioned problems and assume that a representative reference set $\bar{\mathcal{R}}$ generated based on the daily request distribution is in our possession, along with its off-line (near) optimal solution $\bar{\Theta}$. We are interested in how the references ($\bar{\mathcal{R}}$, $\bar{\Theta}$) can be informative and guide the insertion when solving a similar real dynamic problem **LSDARP(**$\mathcal{R}$).

6.1 Preprocessing: Obtain Vehicle Travel Patterns

For each reference route $\bar{\theta} \in \bar{\Theta}$, we compute a specific *travel pattern* $\gamma(\bar{\theta}) \in \Gamma$ during the preprocessing phase. A travel pattern $\gamma(\bar{\theta})$ is a simplified route defined as a list of *pattern points*, where each *pattern point* $\bar{P}$ represents a *cluster* of key points in $\bar{\theta}$. The notion of *key point cluster* is defined as follows:

Definition 6.1 (key point cluster). Given a route $\bar{\theta} = \{\bar{K}_0, \ldots, \bar{K}_i, \ldots, \bar{K}_{M-1}\}$ with M key points, $\{\bar{K}_i, \bar{K}_{i+1}, \ldots, \bar{K}_{i+m}\}$ is *a key point cluster* if and only if:

$$t(\bar{K}_{i+j} - \bar{K}_{i+j-1}) \leqslant \delta^K, \text{ for } 1 \leqslant i \leqslant m,$$

$$t(\bar{K}_{i-1}, \bar{K}_i) > \delta^K, \text{ for } i > 0,$$

and

$$t(\bar{K}_{i+m}, \bar{K}_{i+m+1}) > \delta^K, \text{ for } i < M - 1 - m,$$

where δ^K is the clustering threshold indicating the maximum travel time from a key point to its successor that are in the same cluster.

For example, in Figure 1 where three partial routes are shown, we see that $\bar{K}_2$, $\bar{K}_3$ and $\bar{K}_4$ represent a key point cluster, so in the travel pattern $\gamma(\bar{\theta})$, they are represented by a pattern point $\bar{P}_2$.

The travel patterns are used in the guided insertion process. In *GIM*, each pattern guides the construction of at most one real route $\theta \in \Theta$. Specifically, if a real key point $K \in \theta$ is created via *GIM* under the guidance of a pattern point $\bar{P} \in \gamma$, then we call K a *child*

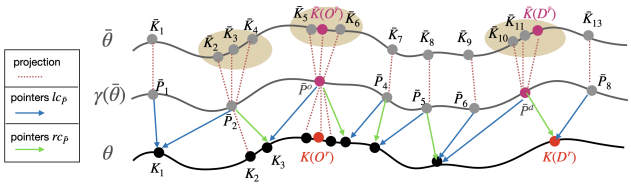

Figure 1: Illustration of the relationships between the reference route, the travel pattern, and the real route

of $\bar{P}$. Any point $\bar{P} \in \gamma$ may project to several *children* in θ. For each point $\bar{P} \in \gamma$, we define two pointers $lc_{\bar{P}}$ and $rc_{\bar{P}}$, where $lc_{\bar{P}}$ points at the preceding key point of the left-most child of $\bar{P}$, and $rc_{\bar{P}}$ points at the right-most child of $\bar{P}$. For example, in Figure 1, $lc_{\bar{P}_2}$ points at K_1, and $lc_{\bar{P}_3}$ points at K_3. For any $\bar{P}$, both $lc_{\bar{P}}$ and $rc_{\bar{P}}$ are initialized as null pointers during the preprocessing process.

6.2 Guided Insertion Process

Given a real request r, *GIM* operates according to the following steps.

6.2.1 Step 1: Retrieve similar reference requests. We first identify from $\bar{\mathcal{R}}$ all the reference requests $\bar{r}$ that are *similar* to r. The *similarity* between requests is defined as follows:

Definition 6.2 (similarity between requests). Two requests r_1 and r_2 are similar if and only if $t(o^{r_1}, o^{r_2}) \leqslant \delta^s$, $t(d^{r_1}, d^{r_2}) \leqslant \delta^s$, and $|e_{O^{r_1}} - e_{O^{r_2}}| \leqslant \delta^t$, where δ^s is a threshold indicating the maximum travel time between two locations, and the threshold δ^t indicates the maximum difference in the earliest pickup time between r_1 and r_2.

If r_1 and r_2 satisfy the above conditions, we use $|e_{O^{r_1}} - e_{O^{r_2}}|$ to define their similarity measure. Let $\bar{\mathcal{R}}^r$ denote the set of retrieved similar reference requests. The set $\bar{\mathcal{R}}^r$ is sorted according to the descending order of their value of similarity measure with r.

6.2.2 Step 2: Construct guide object set. Next, we construct a set of *guide objects*, denoted by GO^r. A *guide object* is a triplet $(\gamma, \bar{P}^o, \bar{P}^d)$ used to guide the guided insertion of r, where $\bar{P}^o$ and $\bar{P}^d$ are two pattern points in the travel pattern γ. For example, as illustrated in Figure 1, a reference request $\bar{r}$ inserted in $\bar{\theta}$ at $\bar{K}(O^r)$ and $\bar{K}(D^r)$ corresponds to the *guide object* $(\gamma(\bar{\theta}), \bar{P}^o, \bar{P}^d)$. Then, GO^r is constructed by sequentially capturing and organizing the corresponding *guide objects* of all the reference requests $\bar{r}$ in alignment with the order specified in $\bar{\mathcal{R}}^r$. We note that it is possible that several reference requests may correspond to the same *guide object*. In GO^r, we only keep one occurrence of the same *guide objects*.

6.2.3 Step 3: Insert r according to the guide object. As mentioned before, the concept of guided insertion is that similar requests should be able to be inserted in the same manner. Given a target request r and the set GO^r, we try to insert r under the guidance of elements in GO^r.

Given $(\gamma, \bar{P}^o, \bar{P}^d) \in GO^r$, we use $\bar{P}^o$ to guide the insertion of O^r, and $\bar{P}^d$ to guide the insertion of D^r.

If γ has not been related to any real route in Θ, then we activate a new vehicle v to service r. Its route θ^v is initialized as two key points K_1, the initial depot, and K_2, the final depot. Next, two key points $K(O^r)$ and $K(D^r)$ supporting the pickup and drop-off services are inserted between K_1 and K_2. Then we relate θ^v to γ by correctly setting the pointers $lc_{\bar{P}}$ and $rc_{\bar{P}}$ for all $\bar{P} \in \gamma$ (see Figure 2).

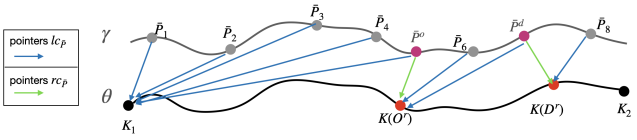

Figure 2: Illustration of the creation of the new route via GIM

If γ is already related to a real route $\theta \in \Theta$, then we utilize $\bar{P}^o$ to construct a list of candidate insertion positions for the pickup service. Specifically, in case $rc_{\bar{P}^o}$ is a *null* pointer (which means that $\bar{P}$ has no related child yet), the candidate list only contains the key point pointed by $lc_{\bar{P}^o}$; otherwise, the list contains all the key points between the two key points pointed by $lc_{\bar{P}^o}$ and $rc_{\bar{P}^o}$. We obtain a candidate list for the drop-off services in the same way. Then, we proceed with a *best-fit* scheme while trying the insertion feasibility of r at the selected candidate key points for the pickup and drop-off services. This means that if at least one feasible insertion is found, we keep the candidates that minimize the insertion cost.

Finally, every time r is inserted in θ at $K(O^r)$ and $K(D^r)$ via *GIM* with the guide object $(\gamma, \bar{P}^o, \bar{P}^d)$, we need to update the pointers $lc_{\bar{P}}$ and $rc_{\bar{P}}$ of pattern points $\bar{P}$ along γ.

We note that the goal of *GIM* when inserting r is to replicate the insertion mode of the most similar reference request. We know that the further forward positioned the guide object GO^r, the more similar the corresponding reference request is to r. Therefore, we implement a *first-fit* scheme to explore GO^r. If r can be inserted under the guidance of $(\gamma, \bar{P}^o, \bar{P}^d)$, we proceed with the insertion, stop the exploration of GO^r, and continue to insert the next request. Otherwise, we explore the next guide object in GO^r.

7 NUMERICAL EXPERIMENTS

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.

7.1 Input Data

We take the transit network of the city of Clermont-Ferrand, France, and its peri-areas. The underlined area contains 13,839 nodes and 31,357 arcs. Among all nodes, 1,469 are selected to be valid pickup and drop-off locations.

The system we study in this paper is still prospective, so there are no available real-life request instances. The instances tested in this paper are self-generated and simulate the intended use cases of the system: providing services to all kinds of travel requests, which are in addition dominated by daily commute demands.

The service period lasts $T = 24$ hours, from 00 : 00 to 24 : 00. We divide the entire period into five time slots: *MS* (Morning Slack, 00:00 ~ 06:00), *MP* (Morning Peak, 06:00 ~ 10:00), *NH* (Normal Hours, 10:00 ~ 15:00), *EP* (Evening Peak, 15:00 ~ 19:00) and *ES* (Evening Slack, 19:00 ~ 24:00). We assume requests to be processed

in one day generally obey the following basic distribution: During *MP*, arond 50% are *typical* that travel from a residential location to a working location. Reversely, *EP* requests follow a symmetric pattern, with half of the requests being *typical* that move from a working position to a residential position. For the requests of *MS*, *NH* and *ES*, their origin and destination are randomly distributed over the network.

To guarantee the representative property of the reference request set $\bar{\mathcal{R}}$ used in *GIM*, we randomly construct it according to the above-introduced basic distribution.

Real requests should globally obey the basic distribution and be "similar" from one day to another while exhibiting some random variations. To simulate such a phenomenon, real request instances are decomposed into two parts: the "random" part and the "similar" part. Requests in the "random" part are randomly generated using the above-defined basic distribution. The "similar" part simulates the stable and recurring pattern of daily requests. We rely on $\bar{\mathcal{R}}$ to generate the corresponding requests. Typically, when generating a request r in this part, we randomly select a reference $\bar{r}$. Then o^r (resp. d^r) is randomly selected among the nodes that are reachable within 3 arcs from $o^{\bar{r}}$ (resp. $d^{\bar{r}}$). And the earliest pickup time e_{O^r} is a random value selected between $e_{O^{\bar{r}}} - 7.5$ minutes and $e_{O^{\bar{r}}} + 7.5$ minutes. In terms of the daily recurring pattern, we consider two scenarios: (*high*): "similar" part accounts for 90% of the requests; and (*moderate*): "similar" part accounts for 50% of the requests.

For each instance, we generate 300, 000 requests. The time length of the pickup window is set at 15 minutes for all requests. The maximum ride time T^r is twice the shortest travel time from o^r to d^r. The load for each request is 1. And as real requests are supposed to be dynamic, we randomly set the submission time t^r_{sub} of r between 0 and e_{O^r}. Requests are submitted on average one hour before e_{O^r}. And in line with the no-rejection assumption, values of t^r_{sub} also satisfy that when processing r at epoch E, we can always activate a vehicle v currently parking at the depot to serve r.

7.2 Analyses of the Effectiveness of GIM

In this work, the static reference problem **LSDARP**($\bar{\mathcal{R}}$) is solved by a best-fit insertion heuristic. The resulting solution $\bar{\Theta}$ contains routes of 1, 621 vehicles. It is worth noting that there are other approximation algorithms (Large Neighborhood Search, Adaptive Large Neighborhood Search, etc.) capable of offering more optimal solutions, but at the expense of significantly increasing the processing time.

The length of each decision epoch e lasts $I_e = 10$ minutes, and for the reason of comparing the efficiency of different approaches, the maximum decision duration τ is not explicitly fixed, and we count the CPU time spent required for each decision epoch. The *GIM* parameters δ^K (key point cluster threshold), δ^s and δ^t (request similarity thresholds) are fixed at 2 minutes, 2 minutes, and 15 minutes, respectively.

In order to test the performance of the *GIM* , we consider three sets of baseline algorithms: **BF** (Best-Fit insertion heuristic), **PFS** (Partial Filtering System), and **FS** (Filtering System). **PFS** and **FS** are two approaches proposed in [14], in which a best-fit insertion heuristic is implemented over the candidate vehicles and insertion positions selected by a filtering system. Specifically, given a request r, in **PFS**, the search space contains all the filtered candidate vehicles and insertion positions; while in **FS**, the search space is further reduced to a subset of selected candidate vehicles along with their candidate insertion positions. The **FS** implemented in this study involves keeping the top 10% best vehicles from the candidate pool. The number of selected vehicles is also bounded between 40 and 140. Then, we can integrate the *GIM* on the upstream side of these three baseline algorithms. Accordingly, we propose three methods: **GIM-BF**, **GIM-PFS** and **GIM-FS**.

Table 1 shows the results of the final fleet size, the total drive time of vehicles, and passengers' average in-vehicle time. The variations represented in percentage are calculated based on the baseline approach **BF**. First, in terms of fleet size, we see that in both scenarios, all the approaches integrated with *GIM* outperform their corresponding baseline approaches. In addition, **GIM-BF** and **GIM-PFS** further reduce the fleet size established by **BF** by almost 6%. Furthermore, thanks to the reduction in the number of used vehicles and the fact that *GIM* helps better organize the routes by following the pre-defined travel patterns, the total drive times of vehicles are also improved. Meanwhile, passenger travel comfort remains the same because better organized routes decrease detours and increase the ride-sharing. All evidence indicates that, by learning from the well-resolved reference solution, *GIM* has the potential to promote vehicle utilization and alleviate emissions. Finally, we also notice that *GIM* is not sensible to similarity scenarios *high* and *moderate*. A possible reason is that, due to the large-scale aspect, we have a great chance of finding a similar reference request, even though it was not deliberately generated as such.

Let us now focus on the scenario *high* to analyze the CPU times spent to process the requests during each decision epoch (see Figure3). First of all, we observe two peaks in the processing time for almost all approaches, corresponding to the high ratio of request submissions during the two peak periods *MP* and *EP*. During most of the epochs corresponding to the time slots *MS, MP, NH* and *ES*, all approaches using *GIM* outperforms their baseline methods. This advantage is especially pronounced during peak hours. During the first few epochs (corresponding to *MS*), methods with *GIM* tend to be less efficient. This is because *GIM* activates numerous vehicles at the beginning of the process, resulting in a larger search space compared to other baseline approaches.

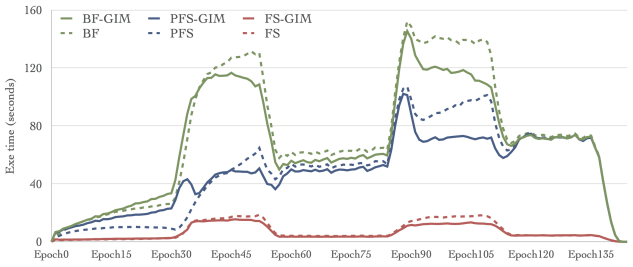

Figure 3: The execution time during each decision epoch

Generally speaking, compared to their baseline approaches, methods integrated with *GIM* showcase the advantages of reducing the execution times during peak hours while providing better routing

Table 1: Results with different approaches under different similarity scenarios with 300k requests

scenario	approach	fleet size	total drive time (h)	average in-vehicle time (min)
high	BF	2,183	25,717.5	16.9
	PFS	2,183 (-0.0%)	-0.0%	16.9
	FS	2,583 (+18.3%)	+22.7%	16.7
	GIM-BF	2,061 (-5.6%)	-4.9%	16.9
	GIM-PFS	2,053 (-6.0%)	-4.9%	16.9
	GIM-FS	2,326 (+6.6%)	+15.8%	16.7
moderate	BF	2,191	25,641.0	17.0
	PFS	2,176 (-0.7%)	+0.1%	17.0
	FS	2,548 (+16.3%)	+23.0%	16.8
	GIM-BF	2,063 (-5.8%)	-4.7%	17.0
	GIM-PFS	2,066 (-5.7%)	-4.7%	17.0
	GIM-FS	2,347 (+7.1%)	+16.8%	16.8

solutions in terms of the fleet size, total drive time and passenger comfort. This highlights the potential of the utilization of *GIM* in solving the dynamic **LSDARP**.

8 CONCLUSION

We introduce a *GIM* upstream of the classic best-fit insertion heuristic. The experiment results show that by learning and imitating a reference static solution, *GIM* stands out in dynamic scenarios by encouraging the fleet to follow optimal vehicle travel patterns. Moreover, its integration with state-of-the-art accelerating algorithms such as the *filtering system* substantially reduces processing time without compromising solution quality. We firmly believe that with a more representative reference problem and a more optimal reference solution, *GIM* should emerge as a highly effective algorithm that is very suitable for addressing dynamic online problems. *GIM* is currently in its early stages as an emerging technology. Our upcoming focus aims to enhance its performance by maximizing the proportion of successful guided insertions within each epoch.

ACKNOWLEDGMENTS

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

REFERENCES

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

[2] Claudia Bongiovanni, Mor Kaspi, and Nikolas Geroliminis. 2019. The electric autonomous dial-a-ride problem. *Transportation Research Part B: Methodological* 122 (April 2019), 436–456. https://doi.org/10.1016/j.trb.2019.03.004

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

[4] Jean-François Cordeau. 2006. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Operations Research* 54, 3 (2006), 573–586. Publisher: INFORMS.

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

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

[7] Masabumi Furuhata, Maged Dessouky, Fernando Ordóñez, Marc-Etienne Brunet, Xiaoqing Wang, and Sven Koenig. 2013. Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological* 57 (Nov. 2013), 28–46. https://doi.org/10.1016/j.trb.2013.08.012

[8] Damianos Gavalas, Charalampos Konstantopoulos, and Grammati E. Pantziou. 2015. Design and Management of Vehicle Sharing Systems: A Survey of Algorithmic Approaches. *CoRR* abs/1510.01158 (2015). arXiv:1510.01158

[9] Timo Gschwind and Michael Drexl. 2019. Adaptive Large Neighborhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem. *Transportation Science* 53, 2 (2019), 480–491. https://doi.org/10.1287/trsc.2018.0837

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

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

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

[13] Xiao Liang, Gonçalo Homem de Almeida Correia, Kun An, and Bart van Arem. 2020. Automated taxis' dial-a-ride problem with ride-sharing considering congestion-based dynamic travel times. *Transportation Research Part C: Emerging Technologies* 112 (March 2020), 260–281. https://doi.org/10.1016/j.trc.2020.01.024

[14] Chijia Liu, Alain Quilliot, Hélène Toussaint, and Dominique Feillet. 2023. A Filtering System for the Large-Scale Dial-A-Ride Problem With Shared Autonomous Vehicles. In *Proceedings of the 12th International Symposium on Information and Communication Technology (SOICT '23)*. Association for Computing Machinery, New York, NY, USA, 679–686. https://doi.org/10.1145/3628797.3628871

[15] Yannik Rist and Michael Forbes. 2021. A New Formulation for the Dial-a-Ride Problem. *Transportation Science* 55 (08 2021). https://doi.org/10.1287/trsc.2021.1044

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

[17] Jiafu Tang, Yuan Kong, Henry Lau, and Andrew W.H. Ip. 2010. A note on "Efficient feasibility testing for dial-a-ride problems". *Operations Research Letters* 38, 5 (2010), 405–407. https://doi.org/10.1016/j.orl.2010.05.002

[18] Yuandong Wang, Hongzhi Yin, Hongxu Chen, Tianyu Wo, Jie Xu, and Kai Zheng. 2019. 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*. ACM, Anchorage AK USA, 1227–1235. https://doi.org/10.1145/3292500.3330877

[19] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. *arXiv:1610.00081 [cs]* (Jan. 2017). arXiv: 1610.00081.