Solving the Dynamic Dial-a-Ride Problem Using a Rolling-Horizon Event-Based Graph

Daniela Gaul
Department of Mathematics, Universität Wuppertal, Germany

Kathrin Klamroth
Department of Mathematics, Universität Wuppertal, Germany

Michael Stiglmayr
Department of Mathematics, Universität Wuppertal, Germany

Abstract
In many ridepooling applications transportation requests arrive throughout the day and have to be answered and integrated into the existing (and operated) vehicle routing. To solve this dynamic dial-a-ride problem we present a rolling-horizon algorithm that dynamically updates the current solution by solving an MILP formulation. The MILP model is based on an event-based graph with nodes representing pick-up and drop-off events associated with feasible user allocations in the vehicles. The proposed solution approach is validated on a set of real-world instances with more than 500 requests. In 99.5% of all iterations the rolling-horizon algorithm returned optimal insertion positions w.r.t. the current schedule in a time-limit of 30 seconds. On average, incoming requests are answered within 2.8 seconds.

2012 ACM Subject Classification Mathematics of computing → Network flows; Applied computing → Multi-criterion optimization and decision-making; Applied computing → Transportation

Keywords and phrases Dial-a-Ride Problem, Ridepooling, Event-Based MILP, Rolling-Horizon, Dynamic Requests

Digital Object Identifier 10.4230/OASIcs.ATMOS.2021.8

Funding Daniela Gaul: This work was partially supported by the state of North Rhine-Westphalia (Germany) within the project “bergisch.smart.mobility”.

Acknowledgements We thank WSW mobil GmbH\textsuperscript{2} for providing dial-a-ride data from their service. Furthermore, we kindly acknowledge three anonymous reviewers for the valuable feedback, which has improved this paper a lot.

1 Introduction

In the dynamic dial-a-ride problem (DARP) a fleet of vehicles must serve transportation requests defined by origin, destination, load and time windows, that arrive throughout the day. An important application are on-demand ridepooling services which are taxi-like services that process transportation requests submitted via a smartphone app. In contrast to taxi-services, where pooling is usually not allowed, customers with similar origin or destination are assigned to the same ride whenever economically and/or ecologically useful. Thus, ridepooling services are a cheap alternative to taxi-services and private cars with the potential to reduce congestion and particulate pollution in big cities. Some prominent

\textsuperscript{1} corresponding author
\textsuperscript{2} www.wsw-online.de

© Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr; licensed under Creative Commons License CC-BY 4.0

Editors: Matthias Müller-Hannemann and Federico Perea; Article No. 8; pp. 8:1–8:16
OpenAccess Series in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
examples are DiDi\(^3\) or UberPool\(^4\). This paper is motivated by Hol-Mich-App\(^5\), a ridepooling service that was recently established in the city of Wuppertal (Germany). In order to achieve a high level of user acceptance in the competition with individual car transport, an efficient and user-friendly route planning is crucial.

Despite being a highly relevant topic of research, the dynamic DARP has been less studied than its static counterpart (see the survey Ho et al. \cite{12}); where it is assumed that all requests are known prior to the start of service. The topic of this paper is the dynamic while still deterministic DARP, i.e., we assume that all information, when received, is known with certainty. An exhaustive and in-depth survey on DARP is given in \cite{12}, and a survey on dynamic pick-up and delivery problems can be found in \cite{3}. Solution strategies to the dynamic DARP are often motivated by the requirement to determine immediately a feasible routing including the new requests. A frequently applied solution strategy to dynamic DARP problems combines two approaches (see e.g. \cite{1, 2, 4, 6, 8, 13, 16, 19, 21, 22}): On the one hand, a new request is inserted using fast and simple insertion heuristics. In the idle time between a pair of new requests, on the other hand, a more complex heuristic or meta-heuristic may be used to continually re-optimize the current solution. We give a brief overview on the variants of insertion and re-optimization heuristics used in the literature.

The first and most simple insertion heuristic tries to insert the new request in the current vehicle routes without relocating already assigned users. If a feasible insertion position is found, the new request is inserted in the best insertion position in terms of incremental cost. Variants of this strategy are employed, for example, by Beaudry et al. \cite{2}, Carotenuto and Martis \cite{6}, Hanne et al. \cite{11}, Häll and Peterson \cite{13}, Lois and Ziliaskopoulos \cite{16}, Madsen et al. \cite{18}, Marković et al. \cite{19}, Psaraftis \cite{20} and Santos and Xavier \cite{21}. The second variant of an insertion heuristic allows the relocation of already assigned users, thus leading two a higher number of possible insertion positions for the new request. For instance, Attanasio et al. \cite{1} use parallel heuristics to solve the dynamic DARP combining random insertion and tabu search. Berbeglia, Cordeau and Laporte \cite{4} run a tabu search heuristic in parallel with a constraint programming algorithm to determine whether a new request can be inserted feasibly in a given solution or not. Luo and Schonfeld \cite{17} relocate requests which are similar w.r.t. time windows and geographic locations whenever a simple insertion heuristic declares a new request to be infeasible. Vallée et al. \cite{22} propose and analyze three different heuristics aiming at reshuffling already accepted requests if a new request’s insertion has been declared infeasible by a service provider’s online system. The heuristics are based on ruin and recreate operators and the ejection chain concept \cite{10}. In \cite{8} new unexpected requests may show up at a vehicle stop. In the idle time between two vehicle stops, a neighborhood of the current vehicle route is created. The insertion of the unexpected request is evaluated for all routes in the neighborhood of the current route. A maximum cluster algorithm that finds for each set of users a maximal subset of users that can be served by one vehicle, developed by Hämé and Hakula \cite{14}, can be used to quickly decide if new requests should be accepted or rejected.

The second phase of a solution approach to the dynamic DARP consists of a reoptimization phase. To improve the current solution in the idle time between a pair of new requests, different variants of local search have been applied. For example, a reinsertion heuristic is used to remove a request from its current route and evaluate the reinsertion of the request into all other routes and/or a swap heuristic exchanges two requests with different routes, see

\[^3\] https://www.didiglobal.com/travel-service/taxi
\[^4\] https://www.uber.com/de/en/ride/uberpool/
\[^5\] https://www.holmich-app.de
e.g. [17, 19, 16, 21]. In [6] the quality of the solution is sought to be improved by reinserting the entire set of accepted requests. In [13] several ruin and recreate heuristics are combined and compared; in particular ruin methods based on the removal of sequences of requests have proven to improve the quality of solutions. Attanasio et al. [1], Beaudry et al. [2] and Berbeglia et al. [4] make use of (different variants) of tabu search in the improvement phase.

Contribution

As highlighted in the literature review, the standard approach to solve the dynamic DARP is to apply a two-phase algorithm consisting of an insertion heuristic and a reoptimization phase. In this paper, we suggest a more global perspective and aim at the iterative computation of exact optimal solutions that satisfy all feasibility constraints and that respect previous routing decisions. Only when this global optimization exceeds a prespecified time limit of 30 seconds without proving global optimality, the computed schedule is reoptimized in the following iteration. We present computational experiments for real-world data from a ridepooling service in the city of Wuppertal in Germany with up to 500 requests. In all tested instances the average response time was never more than 2.9 seconds. Moreover, a reoptimization was necessary in no more than 0.5% of the iterations. In all other iterations the algorithm returned a globally optimal solution w.r.t. the current situation, which can generally not be guaranteed by common two-phase heuristics.

The remainder of this paper is structured as follows. A formal problem description and an outline of the solution strategy applied in this paper is given in Section 2. In Section 3 the concept of an event-based graph is explained and transferred to the dynamic DARP by associating a dynamic event-based graph with each subproblem of DARP. The corresponding MILP model is introduced in Section 4. Finally, the procedure of updating the event-based graph and solving the MILP model, resulting in a decision on the acceptance of new requests, is outlined in the framework of a rolling-horizon algorithm in Section 5. To validate our approach, computational results on two real-world instances are presented in Section 6. A short summary of our results is given in Section 7. A list of parameters and variables used throughout this paper can be found in the appendix.

2 Problem Description

In this paper, we consider a dynamic DARP in which a finite set of n transport requests submitted by users have to be either accepted and scheduled or rejected. The transport service is provided by a fleet of K vehicles with capacity Q. All vehicles are situated at a depot, denoted by 0, when the service is started. Let R := \{1, \ldots, n\} denote the transport requests/users. We consider discrete points in time \(\tau_1 \leq \cdots \leq \tau_n\) such that request \(i\) becomes known at time \(\tau_i - \Delta\), where \(\Delta \geq 0\) is the predefined time-limit for the update of the current solution (we set \(\Delta = 0.5\) minutes in our numerical experiments). Each request \(i \in R\) has an associated pick-up location, denoted by \(i^+\), and an associated drop-off location, referred to as \(i^-\). Let \(P := \{1^+, \ldots, n^+\}\) denote the set of all pick-up locations and let \(D := \{1^-, \ldots, n^-\}\) denote the corresponding set of drop-off locations. Moreover, a number of requested seats \(q_i \geq 1\) and a service time of \(s_i \geq 0\) minutes (needed to enter or leave the vehicle) is associated with each request \(i \in R\). To simplify the notation, we set \(q_i^+ := q_i, s_i^+ := s_i, q_i^- := q_i, s_i^- := s_i\) and \(q_0 := 0\) as well as \(s_0 := 0\). The direct travel time from pick-up location \(i^+\) to drop-off location \(i^-\) of request \(i\) is denoted by \(t_i\). The maximum acceptable ride time of each request \(i \in R\) is bounded from above by \(L_i\). For each request, a pick-up time window \([e_i^+, \ell_i^+]\) is constructed, where the lower bound equals the desired pick-up time specified by the user.
The drop-off time window \([e_i^-, \ell_i^-]\) can be computed from the pick-up time window using \(e_i^- = e_i + s_{i+} + t_i\) and \(\ell_i^- = \ell_i + s_{i+} + L_i\). We assume that there is a fixed duration of service \(T\), resulting in a time window \([e_0, \ell_0]\) associated with the depot, where \(e_0\) denotes the start and \(\ell_0 := e_0 + T\) denotes the end of service. Every user that is accepted is communicated a pick-up time \(\Gamma_i\). This time may not be postponed by more than \(\gamma\) minutes.

Due to the dynamic nature of the problem, at any time \(\tau\) only the requests that have arrived up to time \(\tau\) are known. In addition, some requests might have been rejected and some of the accepted requests might already have been delivered to their drop-off location at time \(\tau\). Therefore, at any time \(\tau\), only a subproblem \(\text{DARP}(\tau)\) related to the active requests \(\mathcal{A}(\tau)\) at time \(\tau\) needs to be considered which comprises all requests that are known but neither rejected nor dropped-off w.r.t. the current solution \(x(\tau)\). To distinguish between these different types of requests at a given time \(\tau\), let

- \(\mathcal{N}(\tau)\) denote the subset of new requests that were revealed at time \(\tau - \Delta\),
- \(\mathcal{S}(\tau)\) denote the subset of scheduled requests, i.e. requests that have been accepted but have not been picked-up up to time \(\tau\),
- \(\mathcal{P}(\tau)\) denote the subset of picked-up requests that have not been dropped-off up to time \(\tau\),
- \(\mathcal{D}(\tau)\) denote the subset of dropped-off requests up to time \(\tau\) and
- \(\mathcal{R}(\tau)\) denote the subset of rejected requests up to time \(\tau\).

Then \(\mathcal{A}(\tau) = \mathcal{N}(\tau) \cup \mathcal{S}(\tau) \cup \mathcal{P}(\tau)\) while \(\mathcal{D}(\tau), \mathcal{R}(\tau) \not\subseteq \mathcal{A}(\tau)\). Note that the sets \(\mathcal{S}(\tau), \mathcal{P}(\tau), \mathcal{D}(\tau), \mathcal{R}(\tau)\) do in fact not only depend on the time \(\tau\) but also on the solutions determined in previous time steps. Each feasible solution \(x(\tau)\) to a subproblem \(\text{DARP}(\tau)\) consists of at most \(K\) vehicle routes which start and end at the depot. If a user is served by a vehicle, the user has to be picked-up and dropped-off by the same vehicle. On the other hand, a rejected user may not be picked-up or dropped-off by any of the vehicles.

A solution to the dynamic DARP is a strategy that, every time one or more new requests are revealed, modifies the solution of the last subproblem so that each of the new requests is either assigned to a vehicle route or rejected. In the course of assigning new requests to already existing vehicle routes, old requests, if not yet picked-up or dropped-off, might have to be reassigned. However, every request, once accepted, has to be served and every request, once rejected, cannot be served by any vehicle in the following subproblems.

The solution approach we propose in this paper is based on an event-based MILP formulation for the static DARP, see [9], which efficiently generates exact solutions to small to medium sized static benchmark problems in a few seconds. The idea of a solution strategy for the dynamic DARP is as follows: 1. An initial solution is obtained by solving the event-based MILP for the requests that are revealed at time \(\tau_1 - \Delta\), which is interpreted as the time when the routes are initialized. 2. When new requests arrive at time \(\tau_i - \Delta, i \geq 2\), the respective users are notified within 30 seconds whether they have been accepted or rejected. Therefore, the vehicle routes up to time \(\tau_i\) are frozen and the set of active requests \(\mathcal{A}(\tau_i)\) is updated. The underlying event-based graph is modified by removing all nodes and arcs corresponding to rejected requests and partially removing nodes and arcs corresponding to dropped-off or picked-up users. Nodes and arcs for the new requests are added to the event-based graph. Then the MILP is updated and solved again.

### 3 Event-Based Graph Model for a Rolling-Horizon

The MILP model for the static DARP suggested in [9] is based on the identification of events that represent pick-up or drop-off situations, and of their chronology. It was motivated by the work of Bertsimas et al. [5].
Each event is associated with a $Q$-tuple that represents a feasible allocation of users to a vehicle with capacity $Q$. For example, the tuple $(2^+, 5, 3)$ represents an event where user 2 has just been picked-up by a vehicle with capacity $Q = 3$ and where users 3 and 5 are seated in the vehicle. The first entry of such a $Q$-tuple always contains the information on the last pick-up location ($i^+$) or drop-off location ($i^-$) while all remaining entries of the $Q$-tuple, representing the remaining users in the vehicle, are sorted in descending order of their respective indices (request numbers). Empty seats are identified by zero entries, and the depot is represented by the node $0 := (0, \ldots, 0)$.

While this formulation of DARP usually requires a large number of events (and hence variables in the associated MILP model), its strength is that feasibility constraints can be easily represented by an associated event-based graph $G = (V, A)$. The node set $V$ of $G$ represents all feasible events, and directed edges in $A$ indicate all possible event sequences. Infeasible user allocations can already be identified (and omitted from $V$) when generating events, and directed edges between events are introduced if the corresponding event sequence is feasible. Then, every directed flow, i.e. every directed circuit, in $G$ represents one vehicle’s tour.

In order to extend this concept to the dynamic DARP, we assume that solutions are extended iteratively whenever new requests arrive and introduce a dynamic event-based graph $G(\tau) = (V(\tau), A(\tau))$ for the subproblem DARP($\tau$) at time $\tau$. When new requests are revealed at time $\tau - \Delta$, $i \in \{1, \ldots, n\}$, then the event-based graph $G(\tau_i)$ is updated based on the event-based graph $G(\tau_{i-1})$ and the associated solution $x(\tau_{i-1})$ of the last subproblem: Nodes and arcs corresponding to rejected, dropped-off and picked-up users are (partially) removed from the graph while nodes and arcs corresponding to new requests are added.

The node set $V(\tau)$ represents events which are feasible w.r.t. the vehicle capacity $Q$ and also reflect time window and ride time constraints. More precisely, given requests $i, j \in A(\tau)$, let $f_{i,j}^1, f_{i,j}^2 \in \{0, 1\}$ indicate the feasibility of the paths $j^+ \rightarrow i^+ \rightarrow j^- \rightarrow i^-$ and $j^+ \rightarrow i^+ \rightarrow i^- \rightarrow j^-$, respectively, w.r.t. time window and ride time constraints. By going through all pairs of requests, feasible combinations of users in vehicles (and hence in events in $V(\tau)$) can be easily identified, see [7]. To simplify the notation we set $f_{i,0}^1 = f_{i,0}^2 = f_{i,i}^1 = 1$. We now formally define the node set of $G(\tau)$: The set of nodes representing an event in which a user $i \in A(\tau) \setminus P(\tau)$ is picked up is called the set of pick-up nodes up to time $\tau$ and is given by

$$V_{i^+}(\tau) := \left\{(v_1, v_2, \ldots, v_Q) : v_1 = i^+, v_j \in A(\tau) \cup \{0\} \setminus \{i\}, f_{i,v_j}^1 + f_{i,v_j}^2 \geq 1 \right\}$$

$$\forall j \in \{2, \ldots, Q\}, \ (v_j > v_{j+1} \lor v_{j+1} = 0) \forall j \in \{2, \ldots, Q-1\}, \sum_{j=1}^{Q} q_{v_j} \leq Q \right\}. \tag{1}$$

Similarly, the set of drop-off nodes up to time $\tau$ corresponds to events where a user $i \in A(\tau)$ is dropped off and is given by

$$V_{i^-}(\tau) := \left\{(v_1, v_2, \ldots, v_Q) : v_1 = i^-, v_j \in A(\tau) \cup \{0\} \setminus \{i\}, f_{v_j,i}^1 + f_{i,v_j}^2 \geq 1 \right\}$$

$$\forall j \in \{2, \ldots, Q\}, \ (v_j > v_{j+1} \lor v_{j+1} = 0) \forall j \in \{2, \ldots, Q-1\}, \sum_{j=1}^{Q} q_{v_j} \leq Q \right\}. \tag{2}$$

We emphasize that one unique (pick-up or drop-off) location is associated with each event through the identification of the user that is picked up or dropped-off in this particular event. Note also that from the set of all pick-up and drop-off nodes associated with an accepted
user, exactly one pick-up and one drop-off node are contained in the dicycle flow representing the vehicle tour to which the user is assigned in the current solution. The set of nodes $V_{\mathcal{A}(\tau)}$ corresponding to the set of active requests $\mathcal{A}(\tau)$ is given by

$$V_{\mathcal{A}(\tau)} = V_0 \cup \bigcup_{i \in \mathcal{A}(\tau) \setminus \mathcal{P}(\tau)} V_{i,+}(\tau) \cup \bigcup_{i \in \mathcal{A}(\tau)} V_{i,-}(\tau),$$

where the set $V_0 := \{0\}$ contains only the depot node. Simply put, $V_{\mathcal{A}(\tau)}$ represents the set of nodes that are available at time $\tau$ but have not been reached by any vehicle (yet). This set does not include nodes (and hence events) corresponding to users that have been rejected or dropped-off up to time $\tau$ since $\mathcal{D}(\tau), \mathcal{R}(\tau) \not\subseteq \mathcal{A}(\tau)$. Moreover, pick-up nodes corresponding to users $\mathcal{P}(\tau)$ are not considered since they have already been reached by a vehicle, where the user has been picked-up. Nodes where a pick-up or drop-off has already been realized up to time $\tau$ are referred to as realized nodes. As a consequence, each request that is known at time $\tau - \Delta$ falls in one of the following three categories:

- If $i \in \mathcal{N}(\tau) \cup \mathcal{S}(\tau) \cup \mathcal{R}(\tau)$, then no associated node (event) is realized since request $i$ was either rejected or the scheduled pick-up and drop-off times are larger than $\tau$.
- If $i \in \mathcal{P}(\tau)$, then exactly one associated node (event) is a realized node, which is a pick-up node.
- If $i \in \mathcal{D}(\tau)$, then exactly one associated pick-up node (event) and one associated drop-off node (event) is realized.

Let $V_{\mathcal{D}(\tau)}$ denote the set of all realized pick-up and drop-off nodes for each user $i \in \mathcal{D}(\tau)$ and let $V_{\mathcal{P}(\tau)}$ denote the set of all realized pick-up nodes associated with each user $i \in \mathcal{P}(\tau)$. Then the node set $V(\tau)$ is defined as

$$V(\tau) := V_{\mathcal{A}(\tau)} \cup V_{\mathcal{D}(\tau)} \cup V_{\mathcal{P}(\tau)}.$$

Hence, for a user $i \in \mathcal{D}(\tau)$ that has been dropped-off up to time $\tau$ only the unique realized pick-up and drop-off nodes are contained in $V(\tau)$, i.e., $V_{i,+}(\tau) := \{v \in V_{\mathcal{D}(\tau)}^{\text{realized}} : v_1 = i^+\}$ and $V_{i,-}(\tau) := \{v \in V_{\mathcal{D}(\tau)}^{\text{realized}} : v_1 = i^-\}$. Analoguously, for a picked-up user $i \in \mathcal{P}(\tau)$ only the unique realized pick-up node is contained in $V(\tau)$, i.e., $V_{i,+}(\tau) := \{v \in V_{\mathcal{P}(\tau)}^{\text{realized}} : v_1 = i^+\}$.

Similar to the node set $V(\tau)$, the arc set $\mathcal{A}(\tau)$ of $G(\tau)$ has to reflect the fact that some routing decisions have already been fixed up to time $\tau$ in the rolling-horizon framework. This motivates the introduction of the concept of realized arcs: Each realized pick-up and drop-off node $v \in V_{\mathcal{D}(\tau)}^{\text{realized}} \cup V_{\mathcal{P}(\tau)}^{\text{realized}}$ is contained in a dicycle flow representing a vehicle’s tour. The incoming arc of a realized node, which is part of this dicycle flow, is referred to as realized arc. We denote the set of realized arcs by $A_{\text{realized}}(\tau)$. Let $v \in V_{\mathcal{D}(\tau)}^{\text{realized}} \cup V_{\mathcal{P}(\tau)}^{\text{realized}}$ be chosen such that there is no arc $a = (v, w) \in A_{\text{realized}}(\tau)$.

$$V(\tau) := V_{\mathcal{A}(\tau)} \cup V_{\mathcal{D}(\tau)} \cup V_{\mathcal{P}(\tau)}.$$

As in the static case, c.f. [9], $A(\tau)$ represents the set of transits from one event node to another. Let $i$ and $j$ be requests that have been revealed up to time $\tau - \Delta$. Then the six subsets $A_k(\tau)$, $k = 1, \ldots, 6$ are defined as follows:
The first set \( A_1(\tau) \) describes the transit from a pick-up node from a set \( V_{i+}(\tau) \) to a drop-off node from a set \( V_{j-} \):

\[
A_1(\tau) := \left\{ \left( (i^+, v_2, \ldots, v_Q), (j^-, w_2, \ldots, w_Q) \right) \in (V_{A(\tau)} \cup V^{l-\text{realized}}(\tau)) \times V_{A(\tau)}: \{j, w_2, \ldots, w_Q\} = \{i, v_2, \ldots, v_Q\} \right\}.
\]

The transit from a pick-up node from a set \( V_{i+}(\tau) \) to another pick-up node from a set \( V_{j+}(\tau) \) with \( j \neq i \) is represented by the following set:

\[
A_2(\tau) := \left\{ \left( (i^+, v_2, \ldots, v_Q), (j^+, w_2, \ldots, w_Q) \right) \in (V_{A(\tau)} \cup V^{l-\text{realized}}(\tau)) \times V_{A(\tau)}: \{i, v_2, \ldots, v_Q-1\} = \{w_2, \ldots, w_Q\} \right\}.
\]

\( A_3(\tau) \) is comprised of arcs which describe the transit from a drop-off node in a set \( V_{i-}(\tau) \) to a pick-up node in a set \( V_{j+}(\tau) \), \( j \neq i \):

\[
A_3(\tau) := \left\{ \left( (i^-, v_2, \ldots, v_Q), (j^+, v_2, \ldots, v_Q) \right) \in (V_{A(\tau)} \cup V^{l-\text{realized}}(\tau)) \times V_{A(\tau)}: i \neq j \right\}.
\]

The transit from a drop-off node from a set \( V_{i-}(\tau) \) to another drop-off node from a set \( V_{j-}(\tau) \), \( j \neq i \), is represented by:

\[
A_4(\tau) := \left\{ \left( (i^-, v_2, \ldots, v_Q), (j^-, w_2, \ldots, w_Q-1, 0) \right) \in (V_{A(\tau)} \cup V^{l-\text{realized}}(\tau)) \times V_{A(\tau)}: \{v_2, \ldots, v_Q\} = \{j, w_2, \ldots, w_Q-1\} \right\}.
\]

A dicycle in \( G(\tau) \) representing a vehicle tour always contains an arc describing the transit from the depot to a pick-up node in a set \( V_{i+}(\tau) \), as well as an arc describing the transit from a drop-off node from a set \( V_{j-}(\tau) \) to the depot. The following two sets describe these transitions:

\[
A_5(\tau) := \left\{ ((0, \ldots, 0), (i^+, 0, \ldots, 0)) \in V_0 \times V_{A(\tau)} \right\};
\]

\[
A_6(\tau) := \left\{ ((j^-, 0, \ldots, 0), (0, \ldots, 0)) \in (V_{A(\tau)} \cup V^{l-\text{realized}}(\tau)) \times V_0 \right\}.
\]

**Example 1.** We give an example of the changes in the event-based graph for three requests and one vehicle with capacity \( Q = 3 \). Let \( R = \{1, 2, 3\} \). The request data is as follows:

\[
\begin{array}{cccc}
i & q_i & \tau_i & [v_{i+}, \ell_{i+}] & [v_{i-}, \ell_{i-}] \\
1 & 1 & 5 & [10, 25] & [15, 40] \\
2 & 2 & 15 & [20, 35] & [30, 50] \\
3 & 2 & 45 & [50, 65] & [55, 80] \\
\end{array}
\]

For the sake of clarity, we assume that all requests are accepted. Furthermore, we assume that the remaining parameters (e.g., travel times) allow all variants of routing described in the following, but are omitted in this example.

When the first request is revealed, we have \( A(\tau_1) = N(\tau_1) = \{1\} \) and \( S(\tau_1) = P(\tau_1) = D(\tau_1) = \emptyset \). The initial graph \( G(\tau_1) \) is depicted in Figure 1a. We assume that by the time request 2 is revealed, user 1 has not been picked up yet, i.e., \( N(\tau_2) = \{2\}, S(\tau_2) = \{1\}, A(\tau_2) = \{1, 2\} \) and \( P(\tau_2) = D(\tau_2) = \emptyset \). Therefore, we only have to add additional nodes and arcs induced by request 2 as illustrated in \( G(\tau_2) \) in Figure 1b. According to the time
windows, by the time request 3 is revealed user 1 must have been dropped-off and user 2 must have been picked-up. We assume that user 2 has not been dropped-off yet and that the vehicle tour induced by the current solution is given by the dicycle

\[ C_1 = \{ (0,0,0), (1^+, 0, 0), (2^+, 1, 0), (1^-, 2, 0), (2^-, 0, 0), (0,0,0) \}. \]

Hence, \( N(\tau_3) = 3, A(\tau_3) = \{ 2, 3 \}, P(\tau_3) = \{ 2 \}, D(\tau_3) = \{ 1 \} \) and \( S(\tau_3) = \emptyset \). The corresponding realized nodes are \( V^\text{realized}_P(\tau_3) = \{ 2^+, 1, 0 \} \) and \( V^\text{realized}_D(\tau_3) = \{ (1^+, 0, 0), (1^-, 2, 0) \} \). The set of realized arcs is \( A^\text{realized}_P(\tau_3) = \{ ((0,0,0), (1^+, 0, 0)), ((1^+, 0, 0), (2^+, 1, 0)), ((2^+, 1, 0), (1^-, 2, 0)) \} \) and \( A^\text{realized}_D(\tau_3) = \{ (1^-, 2, 0) \} \). The update of the event-based graph to obtain \( G(\tau_3) \) is illustrated in Figure 1c. Note that there are no nodes \( v \in V(\tau_3) \) that simultaneously contain users 1 (i.e., \( 1^+ \) or \( 1^- \)) and 3 (i.e., \( 3^+ \) or \( 3^- \)) as \( 1 \notin A(\tau_3) \), which means that according to equations (1) and (2) there are no shared nodes. Similarly, the seats requested by users 2 and 3 combined exceed the vehicle capacity of three.

**4 Event-Based MILP for a Rolling-Horizon**

Based upon the event-based graph model we update and solve an MILP problem in a rolling-horizon strategy whenever new requests arrive, that is, at times \( \tau = \tau_j \) for \( j = 1, \ldots, n \). Every subproblem DARP(\( \tau \)) can be modeled as a variant of a minimum cost flow problem with additional constraints in the dynamic event-based graph \( G(\tau) = (V(\tau), A(\tau)) \).

For the MILP formulation of DARP(\( \tau \)) we use the following additional parameters and variables:

Since every node in the dynamic event-based graph \( G(\tau) = (V(\tau), A(\tau)) \) corresponds to a uniquely determined geographical location, we can associate routing costs \( c_a \geq 0 \) and a travel times \( t_a \geq 0 \) with the respective arcs \( a \in A(\tau) \) in \( G(\tau) \). Let \( \delta^\text{in}(v, \tau) \) and \( \delta^\text{out}(v, \tau) \) denote
the set of incoming and outgoing arcs of \( v \), respectively. A solution of DARP(\( \tau \)) is denoted by \( x(\tau) \) and is composed of the following variables: The binary variables \( x_a \) with \( a \in A(\tau) \) are equal to one if and only if arc \( a \in A(\tau) \) is used by a vehicle. A feasible tour of a vehicle is then represented by a dicycle \( C \) in the dynamic event-based graph \( G(\tau) \) such that \( x_a = 1 \) for all \( a \in C \). Note that due to the structure of the event-based graph, the pick-up and drop-off node of any user are contained in the same dicycle \( C \), representing the assignment of the user to the respective vehicle. If a vehicle has reached the last drop-off location in the dicycle representing its route, it will wait at its current location for new requests until it has to start its journey back to the depot to arrive there before the end of service \( \ell_0 \). Since requests might be rejected, we introduce a binary variable \( p_i \) for each \( i \in A(\tau) \setminus P(\tau) \) with \( p_i = 1 \) indicating that request \( i \) is accepted. To model the beginning of service at a node \( v \in V(\tau) \), i.e. the time at which a vehicle arrives at the location represented by \( v \) to pick-up or drop-off passengers, we use continuous variables \( B_v \). The continuous variables \( d_i, i \in A(\tau) \) measure a user’s excess ride time compared to his or her earliest drop-off time.

The parameters \( x_a^{old} \) and \( B_v^{old} \) are used to store the values of the variables \( x_a \) and \( B_v \) from the previous iteration in the rolling-horizon framework. Once a vehicle has departed from a location, we cannot divert it from its next destination (as this brings technical difficulties related to the calculation of distances, see [3]). Also, if an arc has been realized up to time \( \tau \), it has to be included in a dicycle flow in all later subproblems. Therefore, if \( \tau > \tau_i \) then all partial routes up to time \( \tau \) and hence all variables \( x_a \) corresponding to the set

\[
A^{fixed}(\tau) := \{(v, w) \in A(\tau) : x_a^{old} = 1, \tau \geq B_v^{old} - t_{(v,w)}\}
\]

are fixed in the MILP corresponding to the current subproblem DARP(\( \tau \)). The set of realized arcs \( A^{realized}(\tau) \) is a subset of the set of fixed arcs \( A^{fixed}(\tau) \). We set \( A^{fixed}(\tau_1) = \emptyset \). Furthermore, let \( A^{new}(\tau) \) be the set of all arcs that have not been contained in the graph corresponding to the previous subproblem. We have \( A^{new}(\tau_1) = A(\tau_1) \).

For the remainder of this section, let \( j \in \{1, \ldots, n\} \) be arbitrary but fixed. To prepare the MILP formulation of DARP(\( \tau_j \)), we define a set of travel time constraints \( (C_{v,w}(\tau_j)) \) for all \( (v, w) \in A^{new}(\tau_j) \setminus \delta^{out}(0, \tau_j) \):

\[
B_w \geq \max\{B_v, \tau_j\} + s_{v_1} + t_{(v,w)} - M_{v,w}(\tau_j) \cdot (1 - x_{(v,w)}), \quad (C_{v,w}(\tau_j))
\]

where

\[
M_{v,w}(\tau_j) := \left\{ \begin{array}{ll}
\ell_{v_1} - e_{w_1} + s_{v_1} + t_{(v,w)} & \text{if } B_w \geq \tau_j \\
\tau_j - e_{w_1} + s_{v_1} + t_{(v,w)} & \text{otherwise}
\end{array} \right.
\]

is a sufficiently large constant. The constraints \( (C_{v,w}(\tau_j)) \) guarantee that for all arcs \( (v, w) \in A^{new}(\tau_j) \setminus \delta^{out}(0, \tau_j) \) the beginning of service at a node \( w \) is greater than or equal to the earliest departure time at a preceding node \( v \) plus the time needed to travel from node \( v \) to node \( w \). If \( (v, w) \in A^{new}(\tau_j) \setminus \delta^{out}(0, \tau_j) \), then the arc \( (v, w) \) is related to a new request that has been revealed at time \( \tau_j - \Delta \). This implies that travel from \( v \) to \( w \) can start no earlier than \( \max\{B_v, \tau_j\} + s_{v_1} \). Note that in this case constraint \( (C_{v,w}(\tau_j)) \) can be linearized by rewriting it using two constraints where \( \max\{B_v, \tau_j\} \) is once replaced by \( B_v \) and once by \( \tau_j \). We are now ready to formulate the event-based MILP(\( \tau_j \)) for each subproblem DARP(\( \tau_j \)).

**Event-Based MILP(\( \tau_j \)) for a Rolling-Horizon.**

\[
\begin{align*}
\min & \quad \omega_1 \sum_{a \in A(\tau_j)} c_a x_a + \omega_2 \sum_{i \in A(\tau_j) \setminus P(\tau_j)} (1 - p_i) + \omega_3 \sum_{i \in A(\tau_j)} d_i, \\
\text{s.t.} & \quad \sum_{a \in \delta^{in}(v, \tau_j)} x_a - \sum_{a \in \delta^{out}(v, \tau_j)} x_a = 0 \quad \forall v \in V(\tau_j),
\end{align*}
\]

(3a)

(3b)
8:10 Solving the Dynamic Dial-a-Ride Problem

\[ \sum_{a \in \delta^{in}(v, \tau_j)} x_a = p_i \quad \forall i \in A(\tau_j) \setminus P(\tau_j), \] (3c)

\[ \sum_{a \in \delta^{out}(0, \tau_j)} x_a \leq K, \] (3d)

\[ e_0 \leq B_0 \leq \ell_0, \] (3e)

\[ e_{i+} + (\ell_{i+} - e_{i+}) \left( 1 - \sum_{a \in \delta^{in}(v, \tau_j)} x_a \right) \leq B_v \leq \ell_{i+} \quad \forall i \in A(\tau_j) \setminus P(\tau_j), \] (3f)

\[ e_{i-} \leq B_v \leq e_{i+} + L_i + s_i + (\ell_i - e_i) \sum_{a \in \delta^{in}(v, \tau_j)} x_a \quad \forall i \in A(\tau_j), \] (3g)

\[ B_v \leq \ell_{i+} \left( 1 - \sum_{a \in \delta^{in}(v, \tau_j)} x_a \right) + (\Gamma_i + \gamma) \sum_{a \in \delta^{in}(v, \tau_j)} x_a \quad \forall i \in S(\tau_j), \forall v \in V_i(\tau_j), \] (3h)

\[ B_w - B_v - s_i \leq L_i \quad \forall i \in A(\tau_j), \quad v \in V_i(\tau_j), \quad w \in V_i-(\tau_j), \] (3i)

\[ B_w \geq \tau_j + t_{(v,w)}^{x_{(v,w)}} \quad \forall (v, w) \in \delta^{out}(0, \tau_j) \setminus A^{fixed}(\tau_j), \] (3j)

\[ (C_{v,w}(\tau_k)) \quad \forall (v, w) \in A^{new}(\tau_k) \setminus \delta^{out}(0, \tau_k), \forall k = 1, \ldots, j, \] (3k)

\[ d_i \geq B_v - e_i \quad \forall i \in A(\tau_j), \forall v \in V_i-(\tau_j), \] (3l)

\[ p_i = 1 \quad \forall i \in S(\tau_j), \] (3m)

\[ x_{(v,w)} = 1, \quad B_w = B_w^{old} \quad \forall (v, w) \in A^{fixed}(\tau_j), \] (3n)

\[ p_i \in \{0, 1\} \quad \forall i \in A(\tau_j) \setminus P(\tau_j), \quad d_i \geq 0 \quad \forall i \in A(\tau_j), \] (3o)

\[ x_a \in \{0, 1\} \quad \forall a \in A(\tau_j), \quad B_v \geq 0 \quad \forall v \in V(\tau_j). \] (3p)

The objective function (3a) minimizes the total routing cost, the total excess ride time and the number of unaccepted requests, where \( \omega_1, \omega_2, \omega_3 > 0 \) are weighting parameters that can be adapted to represent the respective importance of these optimization criteria. The flow conservation constraints (3b) ensure that only dicycle flows in \( G(\tau_j) \) are feasible. Every accepted user has to be picked-up at one of its pick-up nodes by exactly one vehicle (3c). Constraint (3d) is a capacity constraint on the number of vehicles. The constraints (3e)–(3g) are time-window constraints for the vehicles to arrive at events (nodes). Constraints (3h) guarantee that the start of service at a pick-up node of a user \( i \in S(\tau_j) \) which has not been picked-up yet, is not later than the pick-up time \( \Gamma_i \) communicated to the user plus an additional constant \( \gamma \). Furthermore, the maximum ride time of a user is bounded by constraint (3i), while constraints (3j)–(3k) model the travel-time from node to node. Constraints (3l) measure a user’s excess ride time. The constraints (3m) ensure that a request is contained in a vehicle’s route if and only if it is accepted (indicated by \( p_i = 1 \)). Finally, constraints (3n) ensure that the next solution respects the partial routes up to time \( \tau_j \), including the scheduled service times that are inherited from the previous iteration. Vehicle capacity, pairing and precedence constraints are ensured by the structure of the event-based graph. Furthermore, it guarantees that picked-up users will not be relocated to any other vehicle and that they will eventually be dropped-off. Note that requests that have been accepted but have not been picked-up or dropped-off yet may be assigned to other vehicles in the next iteration.
5 A Rolling-Horizon Algorithm

We now present the essential aspects of the rolling-horizon algorithm. The approach is based on iteratively updating the dynamic event-based graph whenever new requests arrive, given the information obtained from the previous solution. Then the corresponding MILP is resolved. For each new request we have to determine whether it can be feasibly integrated into the existing schedule. If this is possible, then a schedule including the new request that minimizes routing costs and excess ride time is computed. We impose a time limit of 30 seconds to decide how to process new requests. If the solution returned by the MILP solver is not yet known to be optimal due to this time limit, then the solution is reoptimized in the next iteration. Note that this reoptimization can only consider variables that have not yet been fixed due to the advanced time. In the following, let $\delta$ be a timer that ensures this time limit by measuring the time in minutes needed to execute lines 4–8 in Algorithm 1.

\begin{algorithm}
\caption{Rolling-horizon algorithm for dynamic DARP.}
\begin{algorithmic}[1]
\State $(x, B, p, d) = \text{solve}(\text{MILP}(\tau_1))$
\For{$i = 2 \ldots n$} \Comment{new requests $N(\tau_i)$ are revealed}
\State Start timer $\delta = 0$
\State Determine $D(\tau_i), R(\tau_i)$ and $P(\tau_i)$
\State $A(\tau_i) = A(\tau_{i-1}) \cup N(\tau_i) \setminus (D(\tau_i) \cup R(\tau_i))$
\State Compute dynamic event-based graph $G(\tau_i)$
\State Determine set of fixed arcs $A^{\text{fixed}}(\tau_i)$ \Comment{fix partial routes up to $\tau_i$}
\State $(x, B, p, d) = \text{solve}(\text{MILP}(\tau_i))$ and stop prematurely when $\delta = \Delta$
\EndFor
\ForEach{request $i \in N(\tau_i)$}
\If{$p_i = 1$}
\State accept request $i$
\Else
\State reject request $i$
\EndIf
\EndFor
\end{algorithmic}
\end{algorithm}

An initial feasible solution containing the initial requests is obtained by solving MILP($\tau_1$). Every time one or more new requests are revealed at times $\tau_i, i \in \{2, \ldots, n\}$, the set of active requests is updated as $A(\tau_i) = A(\tau_{i-1}) \cup N(\tau_i) \setminus (D(\tau_i) \cup R(\tau_i))$ and the dynamic event-based graph corresponding to the current time $\tau_i$ is computed. Note that we do not have to recompute the whole graph in each iteration: All not realized pick-up and drop-off nodes (up to time $\tau_i$) corresponding to dropped-off and denied users and all not realized pick-up nodes (up to time $\tau_i$) corresponding to picked-up users are removed from the graph together with all incident arcs. On the other hand, new nodes and arcs corresponding to new requests are added to the graph and the MILP is updated accordingly. To assure that vehicle routes computed for the current subproblem DARP($\tau_i$) are consistent with the routes that have been executed up to time $\tau_i - \Delta$, the corresponding variables have to be fixed up to time $\tau_i$ before solving the next subproblem MILP($\tau_i$).

6 Computational Results

In this section we assess the performance of Algorithm 1 based on real data from Hol-Mich-App, a dial-a-ride service in the city of Wuppertal launched in 2020. We use two instances that differ w.r.t. the length of the planning horizon and the number of requests. $Su_8_22$ is an instance with $n = 254$ transportation requests based on accumulated data
Solving the Dynamic Dial-a-Ride Problem

from nine consecutive Sundays in January and February 2021 with service hours from 8 a.m. until 10 p.m., i.e. \( T = 840 \) minutes. \( Sa_6_3 \) consists of \( n = 519 \) requests and is based on accumulated data from nine consecutive Saturdays in January and February 2021 with service hours from 6 a.m. until 3 a.m. the next morning, i.e. \( T = 1260 \) minutes. Note that due to the Covid-19 pandemic the demand for ridepooling services was rather low and hence we accumulated requests to obtain realistic instances. Moreover, the ridepooling cabs which are equipped with six seats were not allowed to transport more than three passengers at a time, i.e., \( Q = 3 \). We used linear regression to approximate unknown travel times from distances and from the known travel times between the pick-up and drop-off locations of the requests. More precisely, the costs \( c_a \) were computed in an OpenStreetMap network of Wuppertal using OSMnx\(^6\), a Python API to OpenStreetMap, and all unknown travel times \( t_a \) were computed from the regression line \( t_a = 1.8246 c_a + 2.369 \). The length of the pick-up time window for each user is 25 minutes, and the lower bound of the pick-up time window is equal to the time when the transportation request was submitted plus the response time of the algorithm, i.e. \( e_i = \tau_i \). Moreover, the maximum ride time of request \( i \) is equal to \( t_i + \max(10, 0.75 t_i) \) minutes. The service time for every request is set to 5 minutes. After some preliminary testing, the parameters in the objective function (3a) are set to \( \omega_1 = 1 \), \( \omega_2 = 60 \) and \( \omega_3 = 0.1 \). Due to the accumulation of request data, we were not given a fixed number of vehicles by the service provider. An evolution of the number of requests during service hours is depicted in Figure 2 in the appendix. In the peak hour, there are 51 requests in instance \( Sa_6_3 \) and 32 requests in instance \( Su_8_22 \). The average length of a direct trip, i.e. driving from pick-up to drop-off location without any additional stops, in both instances is 8.4 minutes. In our tests we evaluate different fleet sizes and solve instance \( Sa_6_3 \) with \( K \in \{12, 14, 16\} \) and instance \( Su_8_22 \) with \( K \in \{6, 8, 10\} \) vehicles. Algorithm 1 was implemented in C++ and all computations were carried out on an Intel Core i7-8700 CPU, 3.20 GHz, 32GB memory using CPLEX 12.10. The computational results can be found in Table 1. For all instances we report the following average values per accepted request: the routing costs (C), the excess ride time in minutes (E), the waiting time from the time of submitting the request until the time of pick-up in minutes (W), the trip length in minutes (TL), the average time to answer a new request in seconds (A), the percentage of requests that are rejected (R), and the number of times CPLEX was terminated prematurely due to a timeout (CT). Furthermore, we listed the average detour factor (DF), the mean occupancy (MO), the percentage of empty mileage (EM) and the system efficiency (SE), which are measures to evaluate the operational efficiency of ridepooling systems. The computation is based on [15] and can be found in Section C.

The results confirm that Algorithm 1 can quickly answer and schedule new requests. No CPLEX timeouts occured in any run of a \( Su_8_22 \) instance. Thus, all 254 requests are either inserted optimally in the given schedule, given the solution of the preceding iteration, or they are rejected due to infeasibility or unacceptable costs. For the larger \( Sa_6_3 \) instances very few timeouts occured, and CPLEX terminated prematurely only one or two times out of the 404 iterations\(^7\). This affected the insertion of five out of 519 requests. The relative MIP gap in these iterations ranged from 0.4% to 0.5%. Moreover, a reoptimization was necessary only in

\(^6\) https://github.com/gboeing/osmnx

\(^7\) There are less than 519 iterations since several requests are revealed at the same time.
Table 1. Computational results for instances from Hol-mich-App.

<table>
<thead>
<tr>
<th>Instance</th>
<th>K</th>
<th>C</th>
<th>E</th>
<th>W</th>
<th>TL</th>
<th>DF</th>
<th>MO</th>
<th>EM</th>
<th>SE</th>
<th>A</th>
<th>R</th>
<th>CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sa_6_3</td>
<td>12</td>
<td>4.4</td>
<td>12.2</td>
<td>9.7</td>
<td>11.6</td>
<td>1.1</td>
<td>1.6</td>
<td>0.3</td>
<td>1.0</td>
<td>2.9</td>
<td>3.5</td>
<td>1</td>
</tr>
<tr>
<td>Sa_6_3</td>
<td>14</td>
<td>4.4</td>
<td>12.2</td>
<td>9.6</td>
<td>11.7</td>
<td>1.1</td>
<td>1.6</td>
<td>0.3</td>
<td>1.0</td>
<td>2.8</td>
<td>3.3</td>
<td>2</td>
</tr>
<tr>
<td>Sa_6_3</td>
<td>16</td>
<td>4.4</td>
<td>11.8</td>
<td>9.4</td>
<td>11.5</td>
<td>1.1</td>
<td>1.5</td>
<td>0.3</td>
<td>1.0</td>
<td>2.7</td>
<td>3.1</td>
<td>1</td>
</tr>
<tr>
<td>Su_8_22</td>
<td>6</td>
<td>4.7</td>
<td>15.9</td>
<td>13.0</td>
<td>12.0</td>
<td>1.2</td>
<td>1.5</td>
<td>0.3</td>
<td>0.9</td>
<td>0.5</td>
<td>3.5</td>
<td>0</td>
</tr>
<tr>
<td>Su_8_22</td>
<td>8</td>
<td>4.6</td>
<td>12.3</td>
<td>10.0</td>
<td>11.5</td>
<td>1.1</td>
<td>1.5</td>
<td>0.3</td>
<td>0.9</td>
<td>0.4</td>
<td>1.6</td>
<td>0</td>
</tr>
<tr>
<td>Su_8_22</td>
<td>10</td>
<td>4.6</td>
<td>11.8</td>
<td>9.7</td>
<td>11.4</td>
<td>1.1</td>
<td>1.5</td>
<td>0.3</td>
<td>0.9</td>
<td>0.3</td>
<td>1.6</td>
<td>0</td>
</tr>
</tbody>
</table>

0.5% of the iterations, which implies that only a very low percentage of requests was rejected while there would have been a feasible and profitable insertion position. From comparing the results for Su_8_22 and Sa_6_3 for the different fleet sizes, it becomes evident that by the use of additional vehicles the average routing costs, the average excess ride time, the average waiting time and the average trip length (except Sa_6_3 with K = 14) per accepted user decrease or remain constant. The average detour factor, the mean occupancy, the percentage of empty mileage and the system efficiency remain (nearly) constant for the different values of K, while the percentage of rejected requests decreases with an increasing number of vehicles. The average time to answer new requests ranges from 2.7 to 2.9 seconds (Sa_6_3) and 0.3 to 0.5 seconds (Su_8_22) on average, demonstrating that Algorithm 1 is stable under different vehicle configurations.

7 Conclusions

We present a rolling-horizon approach for the solution of the dynamic dial-a-ride-problem that is based on adaptively updating an event-based MILP formulation. Numerical experiments on medium-sized instances from a recently established ridepooling service in the city of Wuppertal confirm the efficiency and reliability of this approach. By adapting the weighting parameters in the objective function, different preferences w.r.t. service cost and customer satisfaction can be implemented. The approach can also be used to assess the quality gain when increasing the fleet size or when changing other parameters in the model.

References

Solving the Dynamic Dial-a-Ride Problem


## A Parameters and Variables

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>number of transport requests</td>
</tr>
<tr>
<td>$R$</td>
<td>set of transport requests</td>
</tr>
<tr>
<td>$i^+, i^-$</td>
<td>pick-up and drop-off location of request $i$</td>
</tr>
<tr>
<td>$P$, $D$</td>
<td>set of pick-up and set of drop-off locations</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>time allowed to communicate an answer to new requests</td>
</tr>
<tr>
<td>$\tau_i - \Delta$</td>
<td>time at which request $i$ is revealed</td>
</tr>
<tr>
<td>$\tau$</td>
<td>current time</td>
</tr>
<tr>
<td>$A(\tau)$</td>
<td>set of active requests for subproblem DARP($\tau$)</td>
</tr>
<tr>
<td>$N(\tau)$</td>
<td>new requests revealed at $\tau - \Delta$</td>
</tr>
<tr>
<td>$S(\tau), P(\tau), D(\tau), R(\tau)$</td>
<td>subsets of $R$ of scheduled, picked-up, dropped-off and rejected requests up to time $\tau$</td>
</tr>
<tr>
<td>$K$</td>
<td>fleet of vehicles</td>
</tr>
<tr>
<td>$Q$</td>
<td>vehicle capacity</td>
</tr>
<tr>
<td>$q_i$</td>
<td>load associated with request $i$</td>
</tr>
<tr>
<td>$s_i$</td>
<td>service duration associated with request $i$</td>
</tr>
<tr>
<td>$[e_j, l_j]$</td>
<td>time window associated with request location $j$</td>
</tr>
<tr>
<td>$T$</td>
<td>maximum duration of service</td>
</tr>
<tr>
<td>$t_i$</td>
<td>direct travel time from pick-up location $i^+$ to drop-off location $i^-$</td>
</tr>
<tr>
<td>$L_i$</td>
<td>maximum ride time associated with request $i$</td>
</tr>
<tr>
<td>$\Gamma_i$</td>
<td>pick-up time communicated to user $i$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>maximum delay of communicated pick-up time</td>
</tr>
<tr>
<td>$f_{ij}^2, f_{ij}^3$</td>
<td>feasibility of paths $j^+ \rightarrow i^+ \rightarrow j^- \rightarrow i^-$ and $j^+ \rightarrow i^+ \rightarrow i^- \rightarrow j^-$</td>
</tr>
<tr>
<td>$G(\tau) = (V(\tau), A(\tau))$</td>
<td>event-based graph corresponding to subproblem DARP($\tau$)</td>
</tr>
<tr>
<td>$V_{i+}(\tau), V_{i-}(\tau)$</td>
<td>set of pick-up nodes and set of drop-off nodes corresponding to request $i$ and DARP($\tau$)</td>
</tr>
<tr>
<td>$V_{A(\tau)}$, $V_{D(\tau)}$, $V_{realized(\tau)}$</td>
<td>set of nodes corresponding to active requests $A(\tau)$ and DARP($\tau$)</td>
</tr>
<tr>
<td>$V_{last-realized(\tau)}$</td>
<td>set of last realized nodes corresponding to DARP($\tau$)</td>
</tr>
<tr>
<td>$A_{realized}(\tau)$</td>
<td>set of realized arcs corresponding to DARP($\tau$)</td>
</tr>
<tr>
<td>$A_{fixed}(\tau)$</td>
<td>set of fixed arcs corresponding to DARP($\tau$)</td>
</tr>
<tr>
<td>$A_{new}(\tau)$</td>
<td>set of arcs that have not been contained in the arc set of the last subproblem</td>
</tr>
<tr>
<td>$c_a$, $t_a$</td>
<td>routing cost and travel time on arc $a$</td>
</tr>
<tr>
<td>$\delta^{in}(v, \tau), \delta^{out}(v, \tau)$</td>
<td>incoming arcs and outgoing arcs of node $v$ corresponding to DARP($\tau$)</td>
</tr>
<tr>
<td>$x_a^{old}, B_v^{old}$</td>
<td>value of variables $x_a$ and $B_v$ obtained from last subproblem solved</td>
</tr>
<tr>
<td>$\omega_1$, $\omega_2$, $\omega_3$</td>
<td>weighting parameters</td>
</tr>
<tr>
<td>$\delta$</td>
<td>timer in minutes to measure time while executing Algorithm 1</td>
</tr>
</tbody>
</table>
Table 3 List of variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_i$</td>
<td>binary variable indicating if user $i$ is transported or not</td>
</tr>
<tr>
<td>$B_v$</td>
<td>continuous variable indicating the start of service time at node $v$</td>
</tr>
<tr>
<td>$x_a$</td>
<td>binary variable indicating if arc $a$ is used or not</td>
</tr>
<tr>
<td>$d_i$</td>
<td>continuous variable indicating the excess ride time of user $i$ w.r.t. $e_i$</td>
</tr>
</tbody>
</table>

B Additional Data to Computational Results

Figure 2 Evolution of number of requests during service hours.

C Measuring the Operational Efficiency of Ridepooling Systems

The computation of the following efficiency measures are based on [15].

average detour factor $= \frac{\text{passenger kilometers driven}}{\text{passenger kilometers booked}}$

mean occupancy $= \frac{\text{passenger kilometers driven}}{\text{vehicle kilometers occupied}}$

percentage of empty mileage $= \frac{\text{empty mileage}}{\text{total vehicle kilometers}}$

system efficiency $= \frac{\text{mean occupancy} \cdot (1 - \text{percentage of empty mileage})}{\text{average detour factor}}$