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

Daniela Gaul[1]

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-word 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[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

[1] corresponding author

[2] `www.wsw-online.de`

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 reserach, the dynamic DARP has been less studied than its static counterpart (see the survey Ho et al. [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 [12], and a survey on dynamic pick-up and delivery problems can be found in [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. [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. [2], Carotenuto and Martis [6], Hanne et al. [11], Häll and Peterson [13], Lois and Ziliaskopoulos [16], Madsen et al. [18], Marković et al. [19], Psaraftis [20] and Santos and Xavier [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. [1] use parallel heuristics to solve the dynamic DARP combining random insertion and tabu search. Berbeglia, Cordeau and Laporte [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 [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. [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 [10]. In [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 Häme and Hakula [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 $R \coloneqq \{1, \dots, n\}$ denote the transport requests/users. We consider discrete points in time $\tau_1 \leq \dots \leq \tau_n$ 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 $P \coloneqq \{1^+, \dots, n^+\}$ denote the set of all pick-up locations and let $D \coloneqq \{1^-, \dots, n^-\}$ 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 $q_{i^+} = q_{i^-} \coloneqq q_i$, $s_{i^+} = s_{i^-} \coloneqq s_i$ and $q_0 \coloneqq 0$ as well as $s_0 \coloneqq 0$. 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 Γ_i. This time may not be postponed by more than γ minutes.

Due to the dynamic nature of the problem, at any time τ only the requests that have arrived up to time τ 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 τ. Therefore, at any time τ, only a subproblem DARP(τ) related to the *active requests* $\mathcal{A}(\tau)$ at time τ needs to be considered which comprises all requests that are known but neither rejected nor dropped-off w.r.t. the current solution $\boldsymbol{x}(\tau)$. To distinguish between these different types of requests at a given time τ, 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 τ,
- $\mathcal{P}(\tau)$ denote the subset of *picked-up requests* that have not been dropped-off up to time τ,
- $\mathcal{D}(\tau)$ denote the subset of *dropped-off requests* up to time τ and
- $\mathcal{R}(\tau)$ denote the subset of *rejected requests* up to time τ.

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 τ but also on the solutions determined in previous time steps. Each feasible solution $\boldsymbol{x}(\tau)$ to a subproblem DARP(τ) 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 τ_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 capactiy $Q = 3$ 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 $\mathbf{0} \coloneqq (0, \ldots, 0)$.

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 dicycle 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 subprobem DARP(τ) at time τ. When new requests are revealed at time $\tau_i - \Delta$, $i \in \{1, \ldots, n\}$, then the event-based graph $G(\tau_i)$ is updated based on the event-based graph $G(\tau_{i-1})$ and the associated solution $\boldsymbol{x}(\tau_{i-1})$ 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 $i, j \in \mathcal{A}(\tau)$, let $f^1_{i,j}, f^2_{i,j} \in \{0, 1\}$ indicate the feasibility of the paths $j^+ \to i^+ \to j^- \to i^-$ and $j^+ \to i^+ \to i^- \to j^-$, 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 $f^1_{i,0} = f^2_{i,0} = f^1_{0,i} = f^2_{0,i} = 1$. We now formally define the node set of $G(\tau)$: The set of nodes representing an event in which a user $i \in \mathcal{A}(\tau) \setminus \mathcal{P}(\tau)$ is picked up is called the set of *pick-up nodes* up to time τ and is given by

$$V_{i^+}(\tau) \coloneqq \Bigg\{ (v_1, v_2, \ldots, v_Q) \colon v_1 = i^+,\ v_j \in \mathcal{A}(\tau) \cup \{0\} \setminus \{i\},\ f^1_{i,v_j} + f^2_{i,v_j} \geq 1 \\ \forall j \in \{2, \ldots, Q\},\ \big(v_j > v_{j+1} \vee v_{j+1} = 0\big)\, \forall j \in \{2, \ldots, Q-1\},\ \sum_{j=1}^{Q} q_{v_j} \leq Q \Bigg\}. \quad (1)$$

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

$$V_{i^-}(\tau) \coloneqq \Bigg\{ (v_1, v_2, \ldots, v_Q) \colon v_1 = i^-,\ v_j \in \mathcal{A}(\tau) \cup \{0\} \setminus \{i\},\ f^1_{v_j,i} + f^2_{i,v_j} \geq 1 \\ \forall j \in \{2, \ldots, Q\},\ \big(v_j > v_{j+1} \vee v_{j+1} = 0\big)\, \forall j \in \{2, \ldots, Q-1\},\ \sum_{j=1}^{Q} q_{v_j} \leq Q \Bigg\}. \quad (2)$$

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 $V_0 \coloneqq \{\mathbf{0}\}$ contains only the depot node. Simply put, $V_{\mathcal{A}(\tau)}$ represents the set of nodes that are available at time τ 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 τ 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 τ 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 τ.
- 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 $V_{\mathcal{D}(\tau)}^{\text{realized}}$ denote the set of all realized pick-up and drop-off nodes for each user $i \in \mathcal{D}(\tau)$ and let $V_{\mathcal{P}(\tau)}^{\text{realized}}$ 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

$$V(\tau) \coloneqq V_{\mathcal{A}(\tau)} \cup V_{\mathcal{D}(\tau)}^{\text{realized}} \cup V_{\mathcal{P}(\tau)}^{\text{realized}}.$$

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

Similar to the node set $V(\tau)$, the arc set $A(\tau)$ of $G(\tau)$ has to reflect the fact that some routing decisions have already been fixed up to time τ 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 $a = (v, w) \in A^{\text{realized}}(\tau)$. Thus, v is the last realized node in the corresponding dicycle flow at time τ. Such nodes indicate the last realized stop on the current tour, from which on the solution may be modified if this is advantageous given the newly revealed requests. We denote the set of "last realized nodes" as $V^{\text{l-realized}}(\tau)$. Then, the arc set $A(\tau)$ is composed of seven subsets that will be further specified below:

$$A(\tau) = \bigcup_{k=1}^{6} A_k(\tau) \cup A^{\text{realized}}(\tau).$$

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^-}:

$$A_1(\tau) \coloneqq \Big\{\big((i^+, v_2, \ldots, v_Q), (j^-, w_2, \ldots, w_Q)\big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau)\big) \times V_{\mathcal{A}(\tau)} : \{j, w_2, \ldots, w_Q\} = \{i, v_2, \ldots, v_Q\}\Big\}.$$

- 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:

$$A_2(\tau) \coloneqq \Big\{\big((i^+, v_2, \ldots, v_{Q-1}, 0), (j^+, w_2, \ldots, w_Q)\big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau)\big) \times V_{\mathcal{A}(\tau)} : \{i, v_2, \ldots, v_{Q-1}\} = \{w_2, \ldots, w_Q\}\Big\}.$$

- $A_3(\tau)$ 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 $V_{j^+}(\tau)$, $j \neq i$:

$$A_3(\tau) \coloneqq \Big\{\big((i^-, v_2, \ldots, v_Q), (j^+, v_2, \ldots, v_Q)\big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau)\big) \times V_{\mathcal{A}(\tau)} : i \neq j\Big\}.$$

- 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:

$$A_4(\tau) \coloneqq \Big\{\big((i^-, v_2, \ldots, v_Q), (j^-, w_2, \ldots, w_{Q-1}, 0)\big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau)\big) \times V_{\mathcal{A}(\tau)} : \{v_2, \ldots, v_Q\} = \{j, w_2, \ldots, w_{Q-1}\}\Big\}.$$

- 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:

$$A_5(\tau) \coloneqq \Big\{\big((0, \ldots, 0), (i^+, 0, \ldots, 0)\big) \in V_0 \times V_{\mathcal{A}(\tau)}\Big\},$$

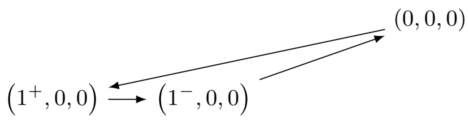
$$A_6(\tau) \coloneqq \Big\{\big((j^-, 0, \ldots, 0), (0, \ldots, 0)\big) \in \big(V_{\mathcal{A}(\tau)} \cup V^{\text{l-realized}}(\tau)\big) \times V_0\Big\}.$$

▶ **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:

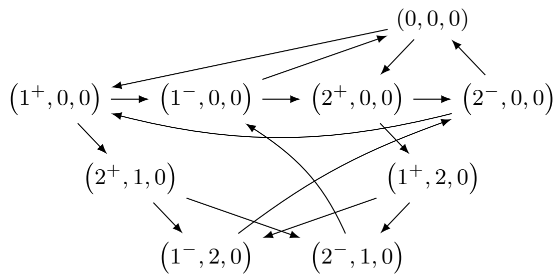
i	q_i	τ_i	$[e_{i^+}, \ell_{i^+}]$	$[e_{i^-}, \ell_{i^-}]$
1	1	5	$[10, 25]$	$[15, 40]$
2	2	15	$[20, 35]$	$[30, 50]$
3	2	45	$[50, 65]$	$[55, 80]$

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.

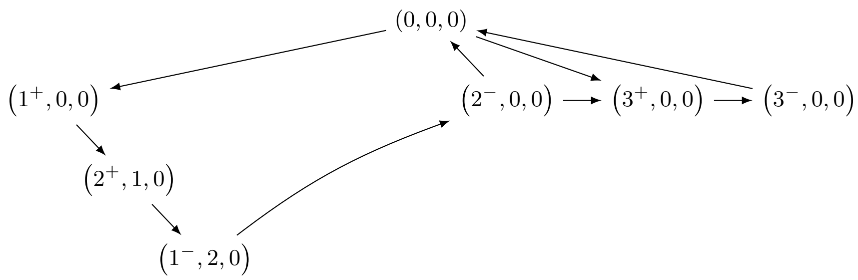
When the first request is revealed, we have $\mathcal{A}(\tau_1) = \mathcal{N}(\tau_1) = \{1\}$ and $\mathcal{S}(\tau_1) = \mathcal{P}(\tau_1) = \mathcal{D}(\tau_1) = \emptyset$. The initial graph $G(\tau_1)$ is depicted in Figure 1a. We assume that by the time request 2 is revealed, user 1 has not been picked-up yet, i.e. $\mathcal{N}(\tau_2) = \{2\}$, $\mathcal{S}(\tau_2) = \{1\}$, $\mathcal{A}(\tau_2) = \{1, 2\}$ and $\mathcal{P}(\tau_2) = \mathcal{D}(\tau_2) = \emptyset$. 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



(a) Graph $G(\tau_1)$.



(b) Graph $G(\tau_2)$.



(c) Graph $G(\tau_3)$.

Figure 1 Evolution of the dynamic event-based graph for an instance with three requests.

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

$$C_1 = \Big\{(0,0,0),\ (1^+,0,0),\ (2^+,1,0),\ (1^-,2,0),\ (2^-,0,0),\ (0,0,0)\Big\}.$$

Hence, $\mathcal{N}(\tau_3) = 3$, $\mathcal{A}(\tau_3) = \{2,3\}$, $\mathcal{P}(\tau_3) = \{2\}$, $\mathcal{D}(\tau_3) = \{1\}$ and $\mathcal{S}(\tau_3) = \emptyset$. The corresponding realized nodes are $V^{\text{realized}}_{\mathcal{P}(\tau_3)} = \{(2^+,1,0)\}$ and $V^{\text{realized}}_{\mathcal{D}(\tau_3)} = \{(1^+,0,0),(1^-,2,0)\}$. The set of realized arcs is $A^{\text{realized}}(\tau_3) = \{((0,0,0),(1^+,0,0)),((1^+,0,0),(2^+,1,0)),((2^+,1,0),(1^-,2,0))\}$ and $V^{\text{l-realized}}(\tau_3) = \{(1^-,2,0)\}$. 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 $1 \notin \mathcal{A}(\tau_3)$, 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 $j = 1,\dots,n$. Every subproblem DARP(τ) 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(τ) 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(τ) is denoted by $\boldsymbol{x}(\tau)$ 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 ℓ_0. Since requests might be rejected, we introduce a binary variable p_i for each $i \in \mathcal{A}(\tau) \setminus \mathcal{P}(\tau)$ 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, $i \in \mathcal{A}(\tau)$ 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 τ, it has to be included in a dicycle flow in all later subproblems. Therefore, if $\tau > \tau_1$ then all partial routes up to time τ and hence all variables x_a corresponding to the set

$$A^{\text{fixed}}(\tau) := \{(v, w) \in A(\tau) \colon x_{(v,w)}^{\text{old}} = 1, \tau \geq B_w^{\text{old}} - t_{(v,w)}\}$$

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

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

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

$$\text{where} \quad M_{v,w}(\tau_j) \geq \begin{cases} \ell_{v_1} - e_{w_1} + s_{v_1} + t_{(v,w)} & \text{if } B_v \geq \tau_j \\ \tau_j - e_{w_1} + s_{v_1} + t_{(v,w)} & \text{otherwise} \end{cases}$$

is a sufficiently large constant. The constraints ($C_{v,w}(\tau_j)$) guarantee that for all arcs $(v, w) \in A^{\text{new}}(\tau_j) \setminus \delta^{\text{out}}(\mathbf{0}, \tau_j)$ 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 $(v, w) \in A^{\text{new}}(\tau_j) \setminus \delta^{\text{out}}(\mathbf{0}, \tau_j)$, 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 τ_j. We are now ready to formulate the event-based MILP(τ_j) for each subproblem DARP(τ_j).

Event-Based MILP(τ_j) for a Rolling-Horizon.

$$\min \omega_1 \sum_{a \in A(\tau_j)} c_a x_a + \omega_2 \sum_{i \in \mathcal{A}(\tau_j) \setminus \mathcal{P}(\tau_j)} (1 - p_i) + \omega_3 \sum_{i \in \mathcal{A}(\tau_j)} d_i, \tag{3a}$$

$$\text{s.t.} \sum_{a \in \delta^{\text{in}}(v, \tau_j)} x_a - \sum_{a \in \delta^{\text{out}}(v, \tau_j)} x_a = 0 \qquad \forall v \in V(\tau_j), \tag{3b}$$

$$\sum_{\substack{a\in\delta^{\text{in}}(v,\tau_j)\\ v\in V_{i^+}}} x_a = p_i \qquad \forall i\in\mathcal{A}(\tau_j)\setminus\mathcal{P}(\tau_j), \tag{3c}$$

$$\sum_{a\in\delta^{\text{out}}(\mathbf{0},\tau_j)} x_a \le K, \tag{3d}$$

$$e_0 \le B_{\mathbf{0}} \le \ell_0, \tag{3e}$$

$$e_{i^+} + (\ell_{i^+} - e_{i^+})\left(1 - \sum_{a\in\delta^{\text{in}}(v,\tau_j)} x_a\right) \le B_v \le \ell_{i^+} \;\; \forall i\in\mathcal{A}(\tau_j)\setminus\mathcal{P}(\tau_j),\, v\in V_{i^+}(\tau_j), \tag{3f}$$

$$e_{i^-} \le B_v \le e_{i^+} + L_i + s_{i^+} + (\ell_{i^+} - e_{i^+}) \sum_{a\in\delta^{\text{in}}(v,\tau_j)} x_a \quad \forall i\in\mathcal{A}(\tau_j),\, v\in V_{i^-}(\tau_j), \tag{3g}$$

$$B_v \le \ell_{i^+}\left(1 - \sum_{a\in\delta^{\text{in}}(v,\tau_j)} x_a\right) + (\Gamma_i + \gamma) \sum_{a\in\delta^{\text{in}}(v,\tau_j)} x_a \qquad \forall i\in\mathcal{S}(\tau_j), \forall v\in V_{i^+}(\tau_j), \tag{3h}$$

$$B_w - B_v - s_{i^+} \le L_i \qquad \forall i\in\mathcal{A}(\tau_j),\, v\in V_{i^+}(\tau_j),\, w\in V_{i^-}(\tau_j), \tag{3i}$$

$$B_w \ge \tau_j + t_{(v,w)} x_{(v,w)} \qquad \forall (v,w)\in\delta^{\text{out}}(\mathbf{0},\tau_j)\setminus A^{\text{fixed}}(\tau_j), \tag{3j}$$

$$(C_{v,w}(\tau_k)) \qquad \forall (v,w)\in A^{\text{new}}(\tau_k)\setminus\delta^{\text{out}}(\mathbf{0},\tau_k),\, \forall k=1,\ldots,j, \tag{3k}$$

$$d_i \ge B_v - e_{i^-} \qquad \forall i\in\mathcal{A}(\tau_j), \forall v\in V_{i^-}(\tau_j), \tag{3l}$$

$$p_i = 1 \quad \forall i\in\mathcal{S}(\tau_j), \tag{3m}$$

$$x_{(v,w)} = 1,\; B_w = B_w^{\text{old}} \qquad \forall (v,w)\in A^{\text{fixed}}(\tau_j), \tag{3n}$$

$$p_i\in\{0,1\} \qquad \forall i\in\mathcal{A}(\tau_j)\setminus\mathcal{P}(\tau_j), \qquad d_i\ge 0 \quad \forall i\in\mathcal{A}(\tau_j), \tag{3o}$$

$$x_a\in\{0,1\} \quad \forall a\in A(\tau_j), \qquad\qquad B_v \ge 0 \quad \forall v\in V(\tau_j). \tag{3p}$$

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 $i \in \mathcal{S}(\tau_j)$ which has not been picked-up yet, is not later than the pick-up time Γ_i communicated to the user plus an additional constant γ. 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 τ_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 guranatees 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 δ be a timer that ensures this time limit by measuring the time in minutes needed to execute lines 4–8 in Algorithm 1.

Algorithm 1 Rolling-horizon algorithm for dynamic DARP.

1 $(x, B, p, d) = \text{solve}(\text{MILP}(\tau_1))$
2 **for** $i = 2 \dots n$ **do** `// new requests` $\mathcal{N}(\tau_i)$ `are revealed`
3   Start timer $\delta = 0$
4   Determine $\mathcal{D}(\tau_i)$, $\mathcal{R}(\tau_i)$ and $\mathcal{P}(\tau_i)$
5     $\mathcal{A}(\tau_i) = \mathcal{A}(\tau_{i-1}) \cup \mathcal{N}(\tau_i) \setminus (\mathcal{D}(\tau_i) \cup \mathcal{R}(\tau_i))$
6   Compute dynamic event-based graph $G(\tau_i)$
7   Determine set of fixed arcs $A^{\text{fixed}}(\tau_i)$ `// fix partial routes up to` τ_i
8     $(x, B, p, d) = \text{solve}(\text{MILP}(\tau_i))$ and stop prematurely when $\delta = \Delta$
9   **foreach** request $i \in \mathcal{N}(\tau_i)$ **do**
10   **if** $p_i = 1$ **then**
11   accept request i
12   **else**
13   reject request i

An initial feasible solution containing the initial requests is obtained by solving $\text{MILP}(\tau_1)$. Every time one or more new requests are revealed at times τ_i, $i \in \{2, \dots, n\}$, the set of active requests is updated as $\mathcal{A}(\tau_i) = \mathcal{A}(\tau_{i-1}) \cup \mathcal{N}(\tau_i) \setminus (\mathcal{D}(\tau_i) \cup \mathcal{R}(\tau_i))$ and the dynamic event-based graph corresponding to the current time τ_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 τ_i) corresponding to dropped-off and denied users and all not realized pick-up nodes (up to time τ_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 $\text{DARP}(\tau_i)$ 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 τ_i before solving the next subproblem $\text{MILP}(\tau_i)$.

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 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 0.75 minutes and the number of requested seats varies from one to three, i.e. $q_i \in \{1, 2, 3\}$. The drop-off time window is computed based on the pick-up time window, the direct travel time, the maximum ride time and the service time. The maximum delay of communicated pick-up time is set to $\gamma = 5$ 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 inacceptable 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.

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

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

1 Andrea Attanasio, Jean-François Cordeau, Gianpaolo Ghiani, and Gilbert Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004. `doi:10.1016/j.parco.2003.12.001`.

2 Alexandre Beaudry, Gilbert Laporte, Teresa Melo, and Stefan Nickel. Dynamic transportation of patients in hospitals. *OR Spectrum*, 32(1):77–107, 2008. `doi:10.1007/s00291-008-0135-6`.

3 Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010. `doi:10.1016/j.ejor.2009.04.024`.

4 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*, 24(3):343–355, 2012. `doi:10.1287/ijoc.1110.0454`.

5 Dimitris Bertsimas, Patrick Jaillet, and Sébastien Martin. Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1):143–162, 2019. `doi:10.1287/opre.2018.1763`.

6 Pasquale Carotenuto and Fabio Martis. A double dynamic fast algorithm to solve multi-vehicle dial a ride problem. *Transportation Research Procedia*, 27:632–639, 2017. `doi:10.1016/j.trpro.2017.12.131`.

7 Jean-François Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006. `doi:10.1287/opre.1060.0283`.

8 Luca Coslovich, Raffaele Pesenti, and Walter Ukovich. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, 175(3):1605–1615, 2006. `doi:10.1016/j.ejor.2005.02.038`.

9 Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr. Event-based MILP models for ride-hailing applications. *arXiv*, 2021. submitted to European Journal of Operations Research. `arXiv:2103.01817`.

10 Fred Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1-3):223–253, 1996. `doi:10.1016/0166-218x(94)00037-e`.

11 Thomas Hanne, Teresa Melo, and Stefan Nickel. Bringing robustness to patient flow management through optimized patient transports in hospitals. *Interfaces*, 39(3):241–255, 2009. `doi:10.1287/inte.1080.0379`.

12 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, 2018. `doi:10.1016/j.trb.2018.02.001`.

13 Carl H. Häll and Anders Peterson. Improving paratransit scheduling using ruin and recreate methods. *Transportation Planning and Technology*, 36(4):377–393, 2013. `doi:10.1080/03081060.2013.798488`.

14 Lauri Häme and Harri Hakula. A maximum cluster algorithm for checking the feasibility of dial-a-ride instances. *Transportation Science*, 49(2):295–310, 2015. `doi:10.1287/trsc.2013.0495`.

15 Christian Liebchen, Martin Lehnert, Christian Mehlert, and Martin Schiefelbusch. Betriebliche Effizienzgrößen für Ridepooling-Systeme. In *Making Connected Mobility Work*, pages 135–150. Springer Fachmedien Wiesbaden, 2021. `doi:10.1007/978-3-658-32266-3_7`.

16 Athanasios Lois and Athanasios Ziliaskopoulos. Online algorithm for dynamic dial a ride problem and its metrics. *Transportation Research Procedia*, 24:377–384, 2017. `doi:10.1016/j.trpro.2017.05.097`.

17 Ying Luo and Paul Schonfeld. Online rejected-reinsertion heuristics for dynamic multivehicle dial-a-ride problem. *Transportation Research Record: Journal of the Transportation Research Board*, 2218(1):59–67, 2011. `doi:10.3141/2218-07`.

18 Oli B. G. Madsen, Hans F. Ravn, and Jens Moberg Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1):193–208, 1995. `doi:10.1007/bf02031946`.

19 Nikola Marković, Rahul Nair, Paul Schonfeld, Elise Miller-Hooks, and Matthew Mohebbi. Optimizing dial-a-ride services in maryland: Benefits of computerized routing and scheduling. *Transportation Research Part C: Emerging Technologies*, 55:156–165, 2015. `doi:10.1016/j.trc.2015.01.011`.

20 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, 1980. `doi:10.1287/trsc.14.2.130`.

21 Douglas O. Santos and Eduardo C. Xavier. Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications*, 42(19):6728–6737, 2015. `doi:10.1016/j.eswa.2015.04.060`.

22 S. Vallee, A. Oulamara, and W. Ramdane Cherif-Khettaf. New online reinsertion approaches for a dynamic dial-a-ride problem. *Journal of Computational Science*, 47:101199, 2020. `doi:10.1016/j.jocs.2020.101199`.

A Parameters and Variables

Table 2 List of parameters.

Parameter	Description
n	number of transport requests
R	set of transport requests
i^+, i^-	pick-up and drop-off location of request i
P, D	set of pick-up and set of drop-off locations
Δ	time allowed to communicate an answer to new requests
$\tau_i - \Delta$	time at which request i is revealed
τ	current time
$\mathcal{A}(\tau)$	set of active requests for subproblem DARP(τ)
$\mathcal{N}(\tau)$	new requests revealed at $\tau - \Delta$
$\mathcal{S}(\tau)$, $\mathcal{P}(\tau)$, $\mathcal{D}(\tau)$, $\mathcal{R}(\tau)$	subsets of R of scheduled, picked-up, dropped-off and rejected requests up to time τ
K	fleet of vehicles
Q	vehicle capacity
q_i	load associated with request i
s_i	service duration associated with request i
$[e_j, \ell_j]$	time window associated with request location j
T	maximum duration of service
t_i	direct travel time from pick-up loaction i^+ to drop-off location i^-
L_i	maximum ride time associated with request i
Γ_i	pick-up time communicated to user i
γ	maximum delay of communicated pick-up time
$f^1_{i,j}$, $f^2_{i,j}$	feasibility of paths $j^+ \to i^+ \to j^- \to i^-$ and $j^+ \to i^+ \to i^- \to j^-$
$G(\tau) = (V(\tau), A(\tau))$	event-based graph corresponding to subproblem DARP(τ)
$V_{i^+}(\tau)$, $V_{i^-}(\tau)$	set of pick-up nodes and set of drop-off nodes corresponding to request i and DARP(τ)
$V_{\mathcal{A}(\tau)}$	set of nodes corresponding to active requests $\mathcal{A}(\tau)$ and DARP(τ)
$V^{\text{realized}}_{\mathcal{D}(\tau)}$, $V^{\text{realized}}_{\mathcal{P}(\tau)}$	set of realized drop-off and set of realized pick-up nodes corresponding to DARP(τ)
$V^{\text{l-realized}}(\tau)$	set of last realized nodes corresponding to DARP(τ)
$A^{\text{realized}}(\tau)$	set of realized arcs corresponding to DARP(τ)
$A^{\text{fixed}}(\tau)$	set of fixed arcs corresponding to DARP(τ)
$A^{\text{new}}(\tau)$	set of arcs that have not been contained in the arc set of the last subproblem
c_a, t_a	routing cost and travel time on arc a
$\delta^{\text{in}}(v,\tau)$, $\delta^{\text{out}}(v,\tau)$	incoming arcs and outgoing arcs of node v corresponding to DARP(τ)
x^{old}_a, B^{old}_v	value of variables x_a and B_v obtained from last subproblem solved
ω_1, ω_2, ω_3	weighting parameters
δ	timer in minutes to measure time while executing Algorithm 1

Table 3 List of variables.

Variable	Description
p_i	binary variable indicating if user i is transported or not
B_v	continuous variable indicating the start of service time at node v
x_a	binary variable indicating if arc a is used or not
d_i	continuous variable indicating the excess ride time of user i w.r.t. e_{i^-}

B Additional Data to Computational Results

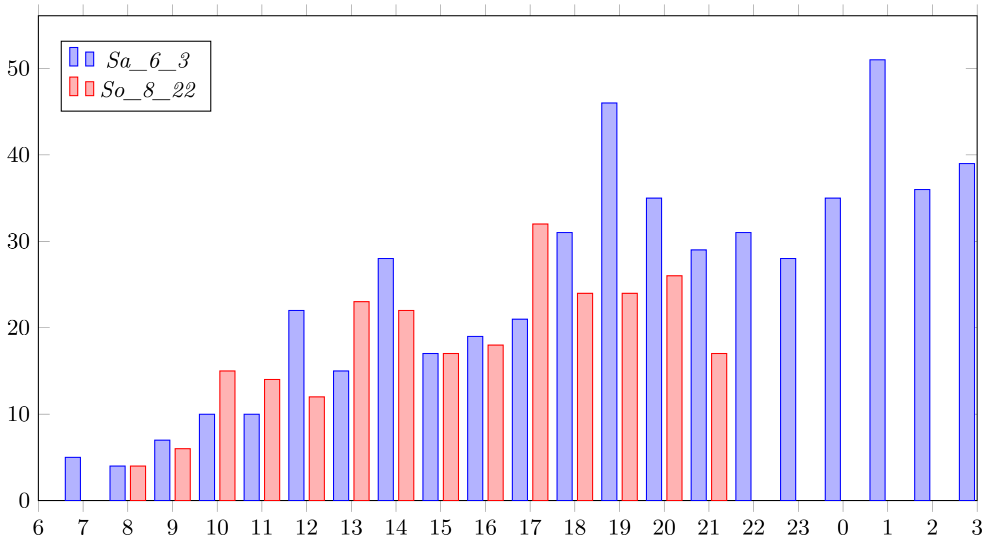


Figure 2 Evolution of number of requests during service hours.

C Measurung the Operational Efficiency of Ridepooling Systems

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

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

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

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

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