The Line-Based Dial-a-Ride Problem with Transfers

Jonas Barth
Department of Computer Science, University of Würzburg, Germany

Kendra Reiter[1]
Department of Computer Science, University of Würzburg, Germany

Marie Schmidt
Department of Computer Science, University of Würzburg, Germany

Abstract

We introduce the *line-based dial-a-ride problem with transfers* (liDARPT), a variation of the well-studied *dial-a-ride problem* (DARP), where vehicles transport requests on-demand but are constrained to operate along a set of lines, and passengers are allowed to transfer between lines on their journey. We develop an event-based solution approach for the liDARPT that relies on the construction of an event-based graph and uses a MILP to find optimal circulations in the event-based graph. To make this solution approach effective, we devise a pre-processing routine to limit the size of the event-based graph. We extensively test our approach on novel benchmark instances, inspired by real-life long-distance bus networks. In our experiments, problem instances with up to 80 requests can be solved to optimality within 15 minutes, and an average of 99.69% of requests are accepted in all instances solved to optimality.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases dial-a-ride, line-based, transfers, on-demand, ridepooling

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.17

Supplementary Material *Software (Source Code)*: `https://github.com/barjon0/liDARPT` [3]
archived at `swh:1:dir:f4c2255f2efac8e14d97cf8b152dd7df1a83841a`

1 Introduction

In many rural areas, public transport is unattractive for potential users for two reasons: due to low demand, scheduled bus services only depart a few times a day, and rural bus lines often visit a large number of stations which leads to a high detour compared to a direct connection between a passenger's origin and destination.

An alternative to organize transport in low-demand areas is to operate transport services fully on-demand, i.e., to plan and schedule routes for each time span, based on the set of individual travel requests known at that time. The corresponding optimization problem is known as the *dial-a-ride problem* (DARP). In the static case, i.e., when the full set of requests is known in advance, the additional flexibility leads to solutions that are much better with respect to the chosen metric. However, the assumption that regular public transport passengers would be willing to decide on, commit to, and send out travel requests (several hours) in advance is questionable.

In this paper, we study a transport system that can be seen an an intermediate option between fully-scheduled and completely on-demand services: in the *line-based dial-a-ride problem with transfers* (liDARPT), the studied transport system is *line-based* in the sense that there is a set of lines that defines admissible vehicle routes (each vehicle is assigned to a line, it may take shortcuts or wait at stations while serving the line, but may only turn when

[1] corresponding author

empty), but uses passenger request data to plan and schedule the specific routes that are realized. Providing this structure is expected to prove more favorable in a dynamic setting, where requests are submitted at or a few minutes before the desired departure time.

As a first step towards exploring the liDARPT as an alternative to existing operational models for public transport, we propose a solution approach for the *static* variant, laying the foundation for future investigations. Being able to solve such a model efficiently allows us to obtain a reference point against which we can compare approaches for the dynamic case.

To solve the liDARPT, we follow an *event-based approach* as first proposed in [12] for the DARP. In a first step, this approach represents the "structural part" of a DARP variant as a so-called *event-based graph*, where each node encodes a distinct *event*, i.e., the boarding or alighting action of a passenger *and* the set of passengers on board of the vehicle during this action. Arcs are added between compatible events, such that feasible vehicle routes constitute circulations. In a second step, a MILP based on the event-based graph is used to find a set of circulations that is temporally feasible and optimizes the objective function. In [26], it was shown that this approach outperforms other MILP-based solution approaches for solving the *line-based dial-a-ride problem* (liDARP), a special case of the liDARPT where there is just one line. However, the tractability of the MILP based on the event-based graph depends crucially on clever pre-processing of the event-based graph, removing events and actions that are infeasible due to spatial or temporal incompatibilities of requests. While effective pre-processing rules for DARP and liDARP have been proposed in [12, 13, 26], there are two factors that complicate building and pre-processing an event-based graph for the liDARPT: a) a passenger may transfer, and thus in consequence may board several vehicles sequentially, and b) the sequence of lines taken by a specific passenger request is not fixed a priori.

The contribution of this paper is fourfold:

- We introduce the *line-based dial-a-ride problem with transfers* (liDARPT) that models the on-demand transportation of passengers in line-based transport systems, explicitly routing passengers through the network and allowing the transfers of passengers between lines.
- We extend the concept of an *event-based graph* to this setting and develop effective non-trivial pre-processing rules to reduce the size of the graph.
- Based on the event-based graph, we state a MILP formulation for the liDARPT that takes into account the routing and transfers.
- We demonstrate the applicability of the developed approach on a set of instances with up to 100 requests, varying the number of lines, number of requests, and the (temporal) request density.

1.1 Related Work

The dial-a-ride problem and variants thereof have been studied extensively in the literature, see the surveys by [9] (up until 2007) and [16] (until 2018) for an overview, as well as the typography provided by [21]. Solution methods for the static setting include exact, MILP-based methods such as branch-and-cut [7, 27, 28] or branch-cut-and-price [15], as well as (meta-)heuristic-based methods, including tabu search [8], simulated annealing [5, 25], and adaptive large neighborhood search [22, 23, 24, 29], to give an overview. Further works study dynamic variants of the DARP, see, e.g., [1, 2, 6, 12, 17].

Recently, a number of DARP variants that allow transfers, named DARPT, have been proposed, see, e.g., [10, 14, 20, 25, 30], whose solution approaches all use (meta-)heuristics. Solely [14], who consider a single transfer station, additionally evaluate a branch-and-cut approach for instances with up to 10 requests.

The line-based dial-a-ride problem (on a single line) was first introduced by [26], wherein different MILP formulations were proposed and compared. Lauerbach et al. [18] study the complexity of the liDARP (and the related MinTurn problem).

In this paper, we develop an event-based approach for the liDARPT, which first constructs an event-based graph encoding structural instance information, and then finds feasible circulations on this graph using a MILP. Such an approach was proposed first for the DARP in [12, 13]. In an event-based graph, nodes represent so-called events, consisting of a boarding or alighting action and information on which passengers are already on board of the vehicle, and arcs represent feasible transitions between the events. In [13], it was already observed that the number of events is exponential in the vehicle capacity for DARP event-based graphs. Event-based MILPs introduce a binary variable for each arc in the event-based graph. To obtain a tractable event-based MILP, it is crucial to delete (nodes and) arcs corresponding to infeasible (events and) transitions. Pre-processing rules for the DARP have been proposed in [11, 13]. In [26], these have been extended for the liDARP, where the additional restrictions on vehicle operations lead to a significant decrease in event-based graph size and computation times for the event-based MILP.

In [11], the authors improve the event-based MILP formulation from [13] to the *location-augmented event-based* (LAEB) formulation by incorporating the time consistency of the location-based (two-index) formulation by Ropke et al. [28], effectively halving the computation times compared to state-of-the-art event-based approaches on tested benchmark instances.

2 Problem Formulation and Model

We summarize all notation in Table 2 in the Appendix.

2.1 Problem Description

In the liDARPT, we are given a set of stations $\mathcal{S}$ and a set of lines $\mathcal{I}$, where each line $i \in \mathcal{I}$ is an ordered sequence of stations λ_i and each station lies on at least one line. The lines may intersect at *transfer stations* $s \in \mathcal{S}^T$. For each line $i \in \mathcal{I}$, the *distance* between two stations $s_1, s_2 \in \lambda_i$ is given by $t(s_1, s_2) > 0$ and respects the triangle inequality. The value $t(s_1, s_2)$ equivalently corresponds to the *length* of the segment between s_1, s_2 on line i. Both distance and length are measured in time.

Each line is assigned at least one vehicle $k \in \mathcal{K}$ which travels according to the driving restrictions introduced by [26] for the liDARP: a vehicle may travel along the line in both directions, may take shortcuts or wait at stations, but may only turn when empty. Vehicles may only serve stations on their respective lines and every vehicle starts and ends its tour at a line-specific depot $s^{\text{depot}} \in \mathcal{S}$. The vehicles assigned to the same line $i \in \mathcal{I}$ are homogeneous in the sense that they have the same capacity $c_i \in \mathbb{N}$.

Furthermore, we are given a set of requests $\mathcal{R}$. Each request $r \in \mathcal{R}$ consists of a number of passengers $q_r \in \mathbb{N}$, a pick-up station $r^+ \in \mathcal{S}$, a drop-off station $r^- \in \mathcal{S}$, a time window for the pick-up and one for the drop-off of the request, denoted as $[e(r^+), l(r^+)]$, $[e(r^-), l(r^-)]$, respectively, and a maximum travel time $L_r \geq 0$ between their initial pick-up (at their origin) and last drop-off (at their destination). We assume each request is willing to wait a fixed amount of time for their pick-up/drop-off, defining the time window length. Each request can either be *accepted*, i.e., it is transported from its origin to destination, or *rejected*. We define a *route option* of a request as an alternating sequence of lines and transfer stations, starting at the origin and ending at the destination, that describes a set of possible options

for transporting this request. Then, a request's *route* consists of a route option together with the vehicle used on each line and the associated pick-up and drop-off times at the transfer points.

At each station, we assume that it takes a fixed total service time $b \geq 0$ to service all requests, i.e., for all requests to leave and enter the vehicle. Requests may transfer from one line to another at a transfer station, but we do not consider transfers between vehicles belonging to the same line.

A solution to the liDARPT then consists of a *plan* for each of the vehicles, and, for each request, a *route* - or the decision to reject the request. A vehicle's *plan* specifies the sequence of stations where the vehicle stops, starting and ending with the depot associated to the vehicle's line, as well as the arrival and departure time at each station. A *route* specifies when and where a passenger boards and alights vehicles.

We have two objectives that we consider in lexicographic order: our primary goal is to maximize the *number of accepted requests*, and the second objective is to minimize the total *traveled distance* by the vehicles, which may reflect both cost and environmental concerns.

2.2 Modeling Route Options of Requests

Each line $i \in \mathcal{I}$ consists of an ordered sequence of stations $\lambda_i := (s_1, \ldots, s_{|\lambda_i|})$ with $\{s_1, \ldots, s_{|\lambda_i|}\} =: \mathcal{S}_i \subseteq \mathcal{S}$ and defines a complete graph on $\mathcal{S}_i$. Then, we denote by *network* the union[2] of all these graphs, which share vertices at exactly the transfer stations $\mathcal{S}^T$. We require that the network is connected (else we consider each connected component separately). Note that this definition gives a *direction* to each line, according to the sequence's order. Each line may be served both in *ascending* and in *descending* order, i.e., requests' route options and vehicles' plans both constitute directed paths in the underlying undirected network.

Since a network of multiple lines may allow for multiple paths between the same stations, a single request may have different *route options*: (unique) paths through the network along one or more lines, from the requests' pick-up to its drop-off location. These route options may differ in length, number of transfers, and number of lines that are used.

Following [20], to represent a route option through the network, we introduce *actions* as follows: for a request $r \in \mathcal{R}$ which transfers between vehicles from/to a line $i \in \mathcal{I}$ at a transfer vertex $s \in \mathcal{S}^T$, we call $\rho^{i,-}_{s,r}$ the *inbound action* and $\rho^{i,+}_{s,r}$ the *outbound action* of r at s on i. That is, the inbound action represents the drop-off (from i) and the outbound action represents the pick-up (to i) of r at s.

Then, a possible route option ϕ^r for a request $r \in \mathcal{R}$ can be formalized as a sequence of (an even number of) actions $(\rho^{i,+}_{r^+,r}, \rho^{i,-}_{s,r}, \rho^{i',+}_{s,r}, \ldots, \rho^{i'',-}_{s',r}, \rho^{i''',+}_{s',r}, \rho^{i''',-}_{r^-,r})$, for $s, s' \in \mathcal{S}^T$, $s \neq s'$, $i, i', i'', i''' \in \mathcal{I}$, where

- the first action is outbound, corresponding to a pick-up at the station r^+,
- all intermediate actions (second to penultimate action) are pairs of inbound and outbound actions at the same transfer station $s \in \mathcal{S}^T$, ensuring transfers,
- the last action is inbound, corresponding to a drop-off at the station r^-, and
- all actions (first to last) are pairs of outbound and inbound actions on the same line $i \in \mathcal{I}$, ensuring consecutiveness in each line.

Note that each station s can only be visited once per route option, forbidding cycles in each requests' travel path.

[2] a *union* of two graphs $G_1 = (V_1, A_1), G_2 = (V_2, A_2)$ is the graph $G := (V_1 \cup V_2, A_1 \cup A_2)$.

We divide this sequence of actions into tuples by pairing each outbound with the following inbound action, i.e., for some $s, s' \in \mathcal{S}^T$, $s \neq s'$, $\phi^r := \Big((\rho^{i,+}_{r^+,r}, \rho^{i,-}_{s,r}), \ldots, (\rho^{i',+}_{s',r}, \rho^{i',-}_{r^-,r})\Big)$, where we call each tuple $(\rho^{i,+}_{s,r}, \rho^{i,-}_{s',r})$ a *split* $\psi^i_{s,s',r}$ of r between stations $s, s' \in \mathcal{S}$, $s \neq s'$, on line $i \in \mathcal{I}$. We construct a sequence of subroutes which are separated exactly by a transfer between two lines at a transfer station. A split may be part of multiple route options.

The *length* of a route option is defined as the sum of lengths of its splits, where the length of a split $\psi^i_{s,s',r}$ is exactly the length $t(s, s')$ on $i \in \mathcal{I}$.

For a request $r \in \mathcal{R}$, we use $\Phi(r)$ to denote the set of all route options and $\Psi(r)$ the set of all splits of r. Further, $\mathcal{P}(r)$ denotes the set of all actions of r and $\mathcal{P}$ denotes the set of all possible actions for all requests.

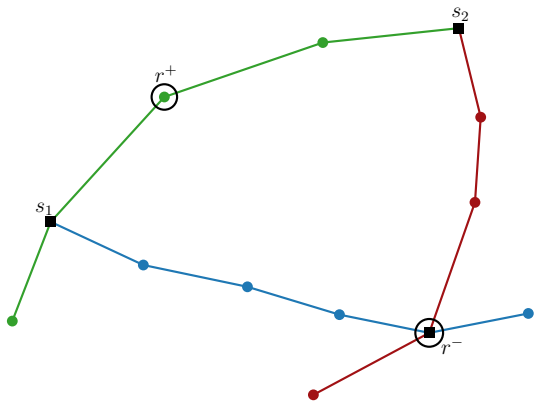


Figure 1 Example of a network with cycles and a single request r with two route options.

An example for a request on a circular network of three lines is pictured in Figure 1. There are two possible route options for request r to travel to their destination: the first route option transfers at station s_1 (along the green, then the blue line), the second route option transfers at s_2 (along the green, then the red line).

We assume that each transfer disrupts the travel experience, as it adds uncertainty to a request's trip and could reduce customer satisfaction. Hence, we impose the following restriction: for each request $r \in \mathcal{R}$, we identify the shortest (wrt. time length) route option from r^+ to r^- and count the amount of transfers. Then, to limit the passenger disutiliy, we only allow route options in $\Phi(r)$ with at most one extra transfer, as transfers are generally perceived as an inconvenience.

The *directionality property* introduced by [26] enforces that a vehicle traveling along a line may only change direction when it is empty. We adopt the same property and introduce a *travel direction*: for each line $i \in \mathcal{I}$ with $\lambda_i = (s_1, s_2, \ldots, s_{|\lambda_i|})$, we define the travel direction from a station s_m to s_n, denoted by $\mathrm{dir}(i, s_m, s_n)$, to be *ascending* if $m < n$, else *descending*. Then, requests traveling on a part of this line, from a transfer station $s \in \mathcal{S}^T$ to $s' \in \mathcal{S}^T$ with $s, s' \in \lambda_i$, inherit the traveling direction $\mathrm{dir}(i, s, s')$ for this split in their route option.

Until here, we have defined route options for requests based only on lines, locations, and actions, without considering when the actions will take place. We now add a timestamp to every action, corresponding to the end of the pick-up/drop-off of this action, and call a timestamped route option a *route* of a request r. Every route needs to be feasible, i.e., it needs to respect the initial pick-up and drop-off time windows of its request, else we can disregard the route and the underlying route option.

Recall that every request $r \in \mathcal{R}$ has a (tight) time window for its pick-up and drop-off, respectively. Using these time windows, along with the service time b and the length of a split, we derive a time window $\left[e(\rho^{i,\pm}_{s,r}), l(\rho^{i,\pm}_{s,r})\right]$ for each action $\rho^{i,\pm}_{s,r}$ associated with r.

In case an action appears in multiple route options, its time window is defined as the union of the individual time window intervals: that is, $e(\rho^{i,\pm}_{s,r})$ is the earliest departure time and $l(\rho^{i,\pm}_{s,r})$ is the latest arrival time among all relevant options.

Similarly, we can then derive time windows of each split from its actions: for a split $\psi^i_{s,s',r}$, we have $e(\psi^i_{s,s',r}) := e(\rho^{i,+}_{s,r})$ and $l(\psi^i_{s,s',r}) := l(\rho^{i,-}_{s',r})$.

2.3 The Event-Based Graph for the liDARPT

In the first step of our event-based approach, we construct an *event-based* graph $G = (\mathcal{V}, \mathcal{A})$, a concept that was proposed first in [13] and adapted for the liDARP in [26]. Here, we discuss how to create an event-based graph for the liDARPT.

An *event* $v \in \mathcal{V}$ is defined as a tuple $v = (\rho^{i,\pm}_{s,r_1}, \psi^i_{s',s'',r_2}, \ldots)$ consisting of an action $\rho^{i,\pm}_{s,r_1}$ and a set of at most $c_i - 1$ splits of requests that may be on board of a vehicle with capacity c_i traveling on the line $i \in \mathcal{I}$ when the action takes place. Each event thus represents a feasible combination of splits which may share part of their journey, in contrast to the events in [13], which consist of combinations of requests. Our representation allows us to incorporate tighter time windows at each transfer station on a requests' route option, and explicitly models the subroutes of each requests' path. Note that each event is associated with a station and a line, namely the station $s \in \mathcal{S}$ and the line $i \in \mathcal{I}$ where the action $\rho^{i,\pm}_{s,r_1}$ takes places.

For the tuple representation of v, we sort the splits in v in ascending order of their request's index. We use $\mathbf{0}_i$ to denote the empty state at the start and end of each vehicle's plan at the depot of its line $i \in \mathcal{I}$. The events constitute the nodes $\mathcal{V}$ of the event-based graph.

For example, the event $(\rho^{i,+}_{s_2,3}, \psi^i_{s_1,s_3,1}, \psi^i_{s_1,s_4,2})$ represents that a vehicle of line i is currently at transfer station s_2 where request 3 boards while requests 1 (who is traveling from station s_1 to s_3) and 2 (traveling from station s_1 to s_4) are already in the vehicle.

The arcs of the event-based graph represent feasible sequences of events. For example, $(\rho^{i,+}_{s_2,3}, \psi^i_{s_1,s_3,1}, \psi^i_{s_1,s_4,2})$ may be connected by an arc to the event $(\rho^{i,-}_{s_3,1}, \psi^i_{s_1,s_4,2}, \psi^i_{s_2,s_4,3})$, where request 1 is dropped-off at transfer station s_3 and requests 2 and 3 are still on board.

The event-based graph G consists of $|\mathcal{I}|$ connected components G_i, one for each line $i \in \mathcal{I}$, where the number of (potential) nodes (and arcs) of each component G_i grows exponentially in the vehicle capacity c_i.

For this reason, we now describe a number of pre-processing steps to identify infeasible combinations of splits, and thus limit the amount of nodes we create. Specifically, our goal is to identify, for every action $\rho^{i,\pm}_{s,r} \in \mathcal{P}(r)$ of every request $r \in R$, the set of *compatible* splits $\psi^i_{g,g',r'} \in \bigcup_{r' \in \mathcal{R}\setminus\{r\}} \Psi(r')$, i.e., splits belonging to requests r' that may be on board of the same vehicle at the time when the action $\rho^{i,\pm}_{s,r}$ is occurring. Note that we only need to consider splits traveling on the same line $i \in \mathcal{I}$.

Let $r \in \mathcal{R}$ and $\psi^i_{s,s',r} = (\rho^{i,+}_{s,r}, \rho^{i,-}_{s',r})$ with $s, s' \in \mathcal{S}$, $s \neq s'$, be a split of r traveling on a line $i \in \mathcal{I}$. We check three conditions to determine if another split $\psi^i_{g,g',r'}$, with $g, g' \in \mathcal{S}$, $g \neq g'$, $r' \in \mathcal{R} \setminus \{r\}$, is compatible to an action in $\psi^i_{s,s',r}$:

First, the *directionality property*: we check if the two splits travel in the same direction, i.e., if $\mathrm{dir}(i, g, g') = \mathrm{dir}(i, s, s')$, as there can never be two splits traveling in opposite directions in the same vehicle at the same time.

We proceed by checking the remaining two conditions for either action in the split $(\rho^{i,+}_{s,r}, \rho^{i,-}_{s',r})$ individually and here describe the process for $\rho^{i,+}_{s,r}$ ($\rho^{i,-}_{s',r}$ follows analogously):

Second, the *spatial overlap*: we check if the action's location s lies on the subsequence of stations $(g, \dots, g') \subseteq \lambda_i$.

Third, the *temporal overlap*: recall that each action is assigned a time window during which it has to occur. Then, we check if

$$[e(\rho^{i,+}_{s,r}), l(\rho^{i,+}_{s,r})] \bigcap [e(\rho^{i,+}_{g,r'}), l(\rho^{i,-}_{g',r'})] \neq \emptyset,$$

i.e., if r and r' may overlap in their time windows.

We call a split satisfying these three conditions *compatible* with the action $\rho^{i,+}_{s,r}$ (analogously $\rho^{i,-}_{s',r}$) and repeat these checks for every request $r \in \mathcal{R}$.

As an example, consider the line i depicted in Figure 2 with a split $\psi^i_{s_3,s_5,r} = (\rho^{i,+}_{s_3,r}, \rho^{i,-}_{s_5,r})$ of request r and four other splits for requests r_1, r_2, r_3, and r_4. Suppose the action $\rho^{i,+}_{s_3,r}$ has a time window of $[0, 15]$, while the split $\psi^i_{s_2,s_4,r_3}$ has a time window of $[80, 120]$ and $\psi^i_{s_3,s_7,s_4}$ has a time window of $[5, 60]$.

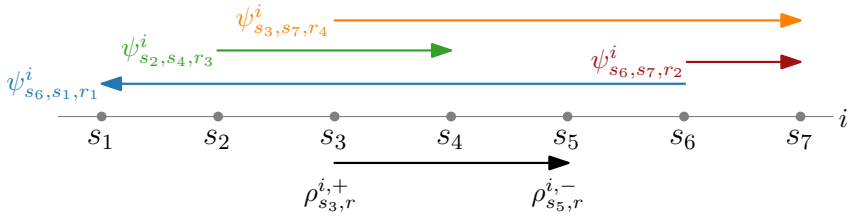


Figure 2 Example of a line i with multiple splits of different requests.

Consider the outbound action $\rho^{i,+}_{s_3,r}$ and check the compatibility of each split in Figure 2:

- split $\psi^i_{s_6,s_1,r_1}$ does not fulfill the directionality property, as it travels in the opposite direction to $\psi^i_{s_3,s_5,r}$, hence it is disregarded.
- split $\psi^i_{s_6,s_7,r_2}$ does not satisfy the spatial overlap since $s_3 \notin (s_6, s_7)$, as its pick-up station s_6 lies beyond the station s_3 in the travel direction, hence it is disregarded.
- split $\psi^i_{s_2,s_4,r_3}$ does not satisfy the temporal overlap since $[0, 15] \cap [80, 120] = \emptyset$, as the earliest start time of its outbound action is later than the time window of $\rho^{i,+}_{s_3,r}$, hence is is disregarded.
- split $\psi^i_{s_3,s_7,r_4}$ fulfills both the directionality property and the spatial overlap as $s_3 \in (s_3, \dots, s_7)$. Additionally, the temporal overlap is satisfied as $[0, 15] \cap [5, 60] = [5, 15] \neq \emptyset$.

Hence, only the split $\psi^i_{s_3,s_7,r_4}$ of request r_4 is compatible to $\rho^{i,+}_{s_3,r}$ in this example.

Once we have identified all compatible splits for each action on a line $i \in \mathcal{I}$, we incrementally build combinations of splits of size at most c_i. In each step, we check if the combination is feasible wrt. the time windows of the contained splits and only explore supersets further if this is guaranteed. See [7] for details on the feasibility checks based on time windows and [4] for further details on the implementation of our method.

We add two further conventions at a transfer station to limit the number of events: if multiple compatible actions take place at the same station $s \in \mathcal{S}^T$, then the inbound actions are handled first. This corresponds to the widely-accepted convention of letting people leave the vehicle first before boarding. Second, if multiple outbound (resp. inbound) actions are compatible and located at the same station $s \in \mathcal{S}^T$, then we only consider the sequence of events in which they board in descending order of their time window end $l(\rho^{i,+}_{s,r})$ (resp. $l(\rho^{i,-}_{s,r})$).

The vertex set $\mathcal{V}$ of our event-based graph G then consists, for every line $i \in \mathcal{I}$, exactly of the feasible combinations $v = (v_1, \ldots, v_m)$ where the first entry v_1 is an action and the remaining entries $v_2, \ldots, v_m$ are splits, with $m \leq c_i$. Each entry corresponds to a different request.

Every event $v = (v_1, \ldots, v_m) \in \mathcal{V}$ is assigned a time window during which it can occur, which is not only dependent on the time window of the action v_1 but also considers the travel time from every predecessor event (for the earliest start time) and the successor events (for the latest end time), similar to the check for temporal overlap.

Lastly, we define the arc set $\mathcal{A}$ of the event-based graph G. We add an arc (v, v') for $v = (\rho^{i,\pm}_{s,r}, v_2, \ldots, v_m)$, $v' = (\rho^{i',\pm}_{s',r'}, v'_2, \ldots, v'_n) \in \mathcal{V}$ of lines $i, i' \in \mathcal{I}$ if the following conditions are met:

- the set of requests in the vehicle after completing the action $\rho^{i,\pm}_{s,r}$ of v is the same as the set of requests in the vehicle before completing the action $\rho^{i',\pm}_{s',r'}$ of v', and
- the events are compatible wrt. their time windows and travel times, i.e.,

$$l(v') \geq \begin{cases} e(v) + t(s, s') + b & \text{if } s \neq s', \\ e(v) & \text{else.} \end{cases}$$

The travel time of the arc $a = (v, v') \in \mathcal{A}$ is given by $t(a) := t(v, v') := t(s, s')$.

Now, each *plan* for a vehicle of line $i \in \mathcal{I}$ corresponds to a circulation on G_i which starts and ends at the vehicle's depot. However, not every circulation is a feasible plan: it has to respect the time windows and maximum travel time of every request. Additionally, the set of plans, jointly, has to ensure that, if a request r is accepted, then all splits of exactly one route option of r are part of the solution. That is, while the described pre-processing in the event-based graph allows to sort out many infeasible combinations of requests, and can handle the logic of vehicle routing well, there are a number of constraints that it cannot represent. This motivates the use of MILP model that we present in Section 2.4.

2.4 Mixed-Integer Linear Programming Model

In this section, we present a MILP model for the liDARPT. It is based on the location-augmented event-based formulation introduced by [11] for the DARP and uses the event-based graph $G = (\mathcal{V}, \mathcal{A})$ we have defined in Section 2.3.

The binary variables x_a, defined for every $a \in \mathcal{A}$, encode which arcs of the event-based are chosen and thus encode the plan for each vehicle, as well as the selected route options for each request. For every request $r \in \mathcal{R}$, the variable p_r indicates if r is accepted.

For the liDARPT, we introduce variables y^r_j for every route option $\phi^r_j \in \Phi(r)$ to denote if this route option is chosen in our solution. This information spans over multiple lines across our underlying network and thus across multiple components of the event-based graph. If a route option is chosen, then all corresponding splits must be part of our solution. At most one route option may be chosen per request, ensuring it is accepted at most once.

To represent the temporal information in our solution, we add a continuous variable $B_{\rho^{i,\pm}_{s,r}}$ for every action $\rho^{i,\pm}_{s,r} \in \mathcal{P}$ to denote the end of this action. As in the LAEB formulation, a single $B_{\rho^{i,\pm}_{s,r}}$ variable is associated with all $v \in \mathcal{V}$ with $v_1 = \rho^{i,\pm}_{s,r}$.

For an event $v \in \mathcal{V}$, we denote its incoming arcs by $\delta^{\text{in}}(v)$ and its outgoing arcs by $\delta^{\text{out}}(v)$. The set $\mathcal{P}(r)^+$ (resp. $\mathcal{P}(r)^-$) contains all outbound (resp. inbound) actions of r, and the set $\mathcal{P}(r)^+_j$ contains all outbound actions belonging to the route option ϕ^r_j of r.

We denote by $\sigma(k)$ the assigned line of a vehicle $k \in \mathcal{K}$ and by $\sigma^{-1}(i)$ the set of vehicles assigned to line $i \in \mathcal{I}$. The earliest departure time (resp. latest arrival time) of a vehicle from the depot of its line $\sigma(k) = i$ is denoted by $e(s_i^{\text{depot}}) = e(\mathbf{0}_i)$ (resp. $l(s_i^{\text{depot}}) = l(\mathbf{0}_i)$).

$$\min \quad \sum_{a \in \mathcal{A}} t(a) \cdot x_a + W \cdot \sum_{r \in R} (1 - p_r) \tag{1a}$$

s.t.

$$\sum_{a \in \delta^{\text{in}}(v)} x_a - \sum_{a \in \delta^{\text{out}}(v)} x_a = 0, \qquad \forall v \in \mathcal{V} \tag{1b}$$

$$\sum_{\substack{a \in \delta^{\text{in}}(v): \\ v_1 \in \mathcal{P}(r)_j^+}} x_a \geq y_j^r, \qquad \forall r \in \mathcal{R}, \phi_j^r \in \Phi(r) \tag{1c}$$

$$\sum_{a \in \delta^{\text{out}}(\mathbf{0}_i)} x_a \leq |\sigma^{-1}(i)|, \qquad \forall i \in \mathcal{I} \tag{1d}$$

$$B_{\rho_{s',r'}^{i',\pm}} \geq B_{\rho_{s,r}^{i,\pm}} + b + t(s,s') \qquad \forall \rho_{s,r}^{i,\pm}, \rho_{s',r'}^{i',\pm} \in \mathcal{P} \colon s \neq s',$$
$$- M_1 \Big(1 - \sum_{\substack{(u,v) \in \mathcal{A}: \\ u_1 = \rho_{s,r}^{i,\pm} \wedge v_1 = \rho_{s',r'}^{i',\pm}}} x_{(u,v)}\Big), \quad \Big((\rho_{s,r}^{i,\pm}, \ldots), (\rho_{s',r'}^{i',\pm}, \ldots)\Big) \in \mathcal{A} \tag{1e}$$

$$B_{\rho_{s,r'}^{i',\pm}} \geq B_{\rho_{s,r}^{i,\pm}} \qquad \forall \rho_{s,r}^{i,\pm}, \rho_{s,r'}^{i',\pm} \in \mathcal{P} \colon$$
$$- M_1 \Big(1 - \sum_{\substack{(u,v) \in \mathcal{A}: \\ u_1 = \rho_{s,r}^{i,\pm} \wedge v_1 = \rho_{s,r'}^{i',\pm}}} x_{(u,v)}\Big), \Big((\rho_{s,r}^{i,\pm}, \ldots), (\rho_{s,r'}^{i',\pm}, \ldots)\Big) \in \mathcal{A} \tag{1f}$$

$$B_{\rho_{r^+,r}^{i,+}} \geq e(\mathbf{0}_i) + b + t\left(\mathbf{0}_i, r^+\right) \sum_{\substack{v \in \mathcal{V}: \\ v_1 = \rho_{r^+,r}^{i,+}}} x_{(\mathbf{0}_i, v)}, \quad \forall r \in \mathcal{R}, i \in \mathcal{I} \colon \rho_{r^+,r}^{i,+} \in \mathcal{P}(r)^+ \tag{1g}$$

$$B_{\rho_{r^-,r}^{i,-}} \leq l(\mathbf{0}_i) - t\left(r^-, \mathbf{0}_i\right) \sum_{\substack{v \in \mathcal{V}: \\ v_1 = \rho_{r^-,r}^{i,-}}} x_{(v, \mathbf{0}_i)}, \quad \forall r \in \mathcal{R}, i \in \mathcal{I} \colon \rho_{r^-,r}^{i,-} \in \mathcal{P}(r)^- \tag{1h}$$

$$e\left(\rho_{s,r}^{i,\pm}\right) \leq B_{\rho_{s,r}^{i,\pm}} \leq l\left(\rho_{s,r}^{i,\pm}\right), \qquad \forall \rho_{s,r}^{i,\pm} \in \mathcal{P} \tag{1i}$$

$$B_{\rho_{r^-,r}^{i',-}} - B_{\rho_{r^+,r}^{i,+}} - b \leq L_r, \qquad \forall r \in R, \phi^r \in \Phi(r) \colon \rho_{r^-,r}^{i',-}, \rho_{r^+,r}^{i,+} \in \phi^r \tag{1j}$$

$$B_{\rho_{s,r}^{i',+}} \geq B_{\rho_{s,r}^{i,-}} - M_2(1 - y_j^r), \qquad \forall r \in \mathcal{R}, s \in \mathcal{S}, \phi_j^r \in \Phi(r) \colon \rho_{s,r}^{i,-}, \rho_{s,r}^{i',+} \in \phi_j^r \tag{1k}$$

$$\sum_{\phi_j^r \in \Phi(r)} y_j^r = p_r, \qquad \forall r \in \mathcal{R} \tag{1l}$$

$$x_a \in \{0,1\}, \qquad \forall a \in \mathcal{A} \tag{1m}$$

$$p_r \in \{0,1\}, \qquad \forall r \in \mathcal{R} \tag{1n}$$

$$y_j^r \in \{0,1\}, \qquad \forall r \in \mathcal{R}, \phi_j^r \in \Phi(r) \tag{1o}$$

The objective function (1a) maximizes the number of accepted requests, adding a large penalty W in case request r is not accepted. As a a secondary goal, the total distance driven by the vehicles (referred to as *traveled distance)* is minimized. Constraints (1b) are the typical flow constraints. Next, constraints (1c) guarantee that, for an accepted request r and selected route option ϕ_j^r, all of the corresponding splits of ϕ_j^r are part of the solution. There may be at most one plan per vehicle for every line i, enforced by constraints (1d).

Constraints (1e) to (1k) handle time constraints. For subsequent events, constraints (1e) and constraints (1f) ensure the end time of the respective actions is consecutive, where constraints (1e) ensure the necessary travel and service time are respected if the events do not occur at the same station. We choose the parameter M_1 large enough to ensure these constraints are only active if the corresponding succession of actions is part of the solution. Then, constraints (1g) guarantee the correctness of the departure times for the very first action on the paths and similarly, constraints (1h) guarantee the correctness of the arrival times for the last action on the paths. Together, constraints (1g) and constraints (1h) ensure each vehicle travels within a pre-specified travel time span. The time windows of each action are ensured by constraints (1i). Furthermore, we make sure the the maximum travel time is upheld with constraints (1j), by checking the difference of the last split's drop-off time and the pick-up time of the first split for a request. For the timing of all subsequent splits of the same request, constraints (1k) enforce that no user is picked up before they even arrive at a station s. Again, we choose M_2 large enough to ensure this constraint is only active if the corresponding route option is chosen. Finally, in constraints (1l), we ensure that exactly one route option is selected if and only if the request is transported.

Note that a split may be part of multiple route options and that it is not forbidden by our MILP that a split may be part of a vehicle's plan, even though none of the corresponding route options are chosen. Hence, after every model solve, we run a clean-up routine, removing unnecessary events from the solution. These events have no effect on the solution: they may not add to the traveled distance nor may they occupy a seat which could otherwise have been assigned to a rejected request, as both would increase our objective function.

We now prove a bound on the objective function penalty W to enforce our lexicographic objective function (1a).

▶ **Lemma 1.** *Let N be sum of lengths of all lines. Setting $W = 2N|\mathcal{R}| + 1$, the optimal solution OPT of (1) accepts the maximum number of requests.*

Proof. Let $W = 2N|\mathcal{R}| + 1$ and assume there exists a different solution ALT with more accepted requests than OPT. We know that the difference in penalty is at least $2N|\mathcal{R}| + 1$. However, we also know that $OPT \leq ALT$. This means that the difference in traveled distance has to be at least $2N|\mathcal{R}| + 1$:

Note that for a given set of accepted requests $|\mathcal{R}'|$, even if all requests need to use all lines from beginning to end and no requests can be transported together due to incompatible time windows, and all requests are accepted, the distance driven to serve them is at most $\sum_{a\in\mathcal{A}} t(a)x_a = 2N|\mathcal{R}'| < 2N|\mathcal{R}| + 1 = W$.

Thus, the improvement in traveled distance cannot compensate for the loss in number of transported passengers. ◀

3 Computational Experiments

In this section, we present numerical experiments for the liDARPT on new, synthetic benchmark instances that are based on real-world bus networks. The code and all instances are available on GitHub[3].

For all presented experiments, every line is assigned two buses of capacity six. The service time b is set to two minutes and we set the maximum waiting time of each request to 15 minutes. The maximum travel time L_r of a request r is given by $t_r^* + 1.2 \cdot \log_{1.2}(t_r^*)$, where t_r^* is the shortest direct travel time of r along the network. We set the penalty weight $W = 3N|\mathcal{R}|$, where N is the sum of lengths of all lines.

[3] `https://github.com/barjon0/liDARPT`

All experiments were conducted on a 24 core Intel i9-13900F machine with 32GB RAM and operating system nixOS 24.11. The model was implemented in Python 3.10 and solved using CPLEX 22.1.1. All runs were limited to 15 minutes of computational time and the results are averaged over three runs. A summary is provided in Table 3 in the Appendix.

3.1 Benchmark Instances

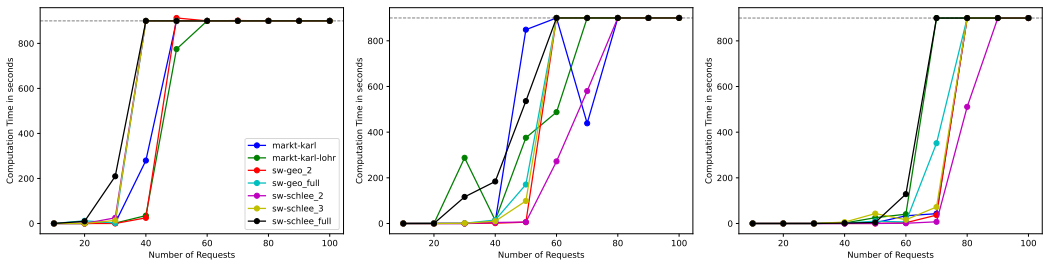
We create a new set of benchmark instances modeled after the long-distance bus network in the region of Unterfranken, Germany. Table 1 provides an overview of the seven networks, where the total transfer degree denotes the sum of degrees of all transfer stations as a further measure of complexity. A visualization of the networks is provided in Figure 12 in the Appendix. Networks with the same base name build upon each other; for example, sw-schlee_3 consists of the sw-schlee_2 network with an additional line.

Table 1 Overview of benchmark networks.

Network	#Lines	#Stops	#Transfer Stops	Total Transfer Deg.	Length [km]
markt-karl	3	15	3	8	109.3
markt-karl-lohr	5	26	5	14	165.6
sw-geo_2	2	21	2	5	76.5
sw-geo_full	4	32	4	13	120.1
sw-schlee_2	2	14	2	6	58.6
sw-schlee_3	3	21	3	11	92.8
sw-schlee_full	5	28	5	17	129.9

For each network, we create multiple instances with 10 to 100 requests distributed over a time span of three, six, or nine hours, where each request r is assigned a pick-up and drop-off station under a uniform random distribution across $\mathcal{S}$. We split the time span into 5-minute intervals and, for each r, draw an earliest pick-up time $e(r^+)$ uniformly at random from the resulting set of intervals. The remaining time bounds, i.e., $l(r^+), e(r^-)$, and $l(r^-)$, are then computed from the maximum wait time of 15 minutes, and the shortest direct and maximum travel times of r. Finally, the number of passengers in a request is chosen by assigning a 90% probability for picking a single passenger, 9% for two, and 1% for three passengers.

3.2 Computational Performance



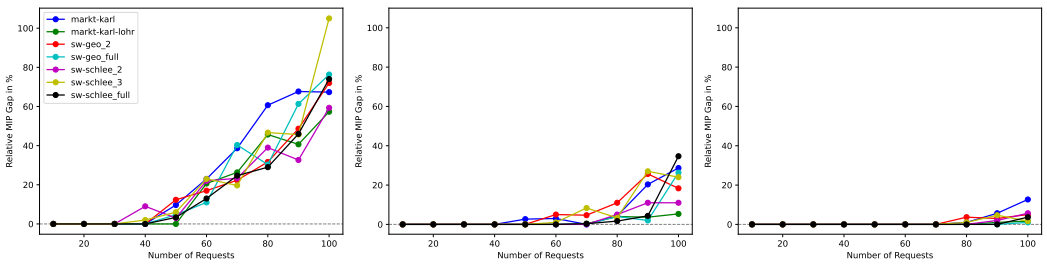
(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 3 Average computation time per instance. The dashed line marks the solver timeout.

Figure 3 visualizes the computation times per instance. As expected, this increases with the number of requests, with timeout first being reached at 40, 50, and 70 requests for time spans of three, six, nine hours, respectively. The computation time seems to correlate stronger with the number of requests than the number of lines of the network. Further, the *temporal density*, defined as the number of requests per hour, has a significant impact: the computation time reaches timeout for a temporal density greater than 13 (for three hours) and 7.77 (for six and nine hours). This may also be explained by the size of the underlying event-based graph, where a higher temporal density correlates to a larger graph, see Figure 5.

Here, the markt-karl-lohr instance with 30 requests in six hours reached a MIP gap of 0.05% within 16 seconds, indicating a near-optimal solution was found early in the optimization process.

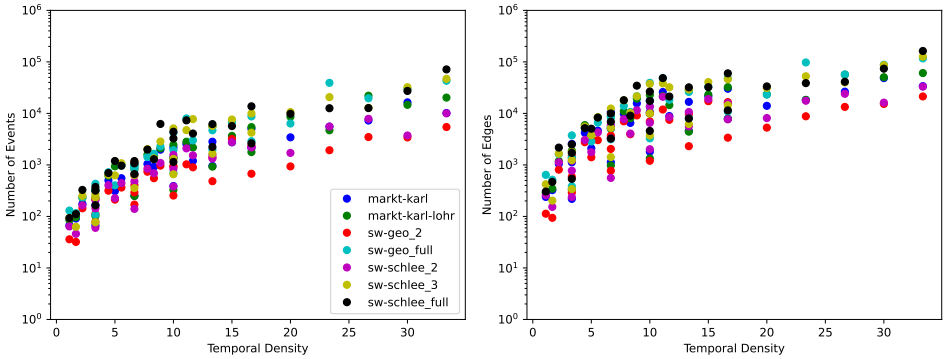
Next, Figure 4 shows the relative MIP gap[4], for all instances. The MIP gap tends to rise with the number of requests for a given time span, and is lower for larger time spans overall. Interestingly, the MIP gap of instances on the supposedly harder networks with more lines, sw-schlee_full and sw-geo_full, is often smaller than on the other networks, rising only for instances with large numbers of requests, e.g., sw-schlee_full at 100 request in six hours.



(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 4 Average relative MIP gap in percentage per instance.

We postulate that this correlates to the number of available vehicles per line: since each line is assigned two vehicles, networks with less lines have a lower seat-to-request ratio, which increases the complexity when aiming to transport all passengers.



(a) Number of nodes. **(b)** Number of edges.

Figure 5 Number of nodes and edges in the underlying event-based graph by temporal density.

[4] the relative gap between the current objective value and the best known bound, in percent.

Figure 5 examines the size of the event-based graph versus the temporal density. For instances of the same temporal density, networks with less lines, such as the sw-geo_2 and sw-schlee_2, produce smaller graphs (with both less nodes and less arcs) than networks with more lines. Interestingly, one of the largest networks, sw-schlee_full is surpassed by sw-schlee_3 (a subgraph with two less lines) multiple times. This suggests that the length of the lines, as well as the number and location of transfer stations, may have an additional impact on the size of the event-based graph.

Finally, Figure 7 shows the number of variables and constraints in the MILP. Both rise with the temporal density and the number of lines in the network: the smallest networks, sw-geo_2 and sw-schlee_2, require both the smallest number of variables and constraints. Larger networks enable more transfers, which in turn also allows for more route options per request, requiring more variables and constraints in our MILP.

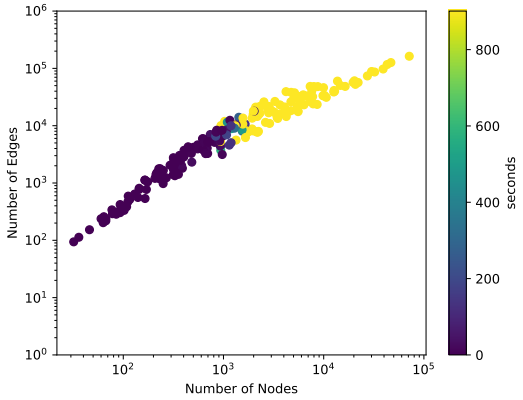


Figure 6 Size of event-based graph in relation to computation time in seconds.

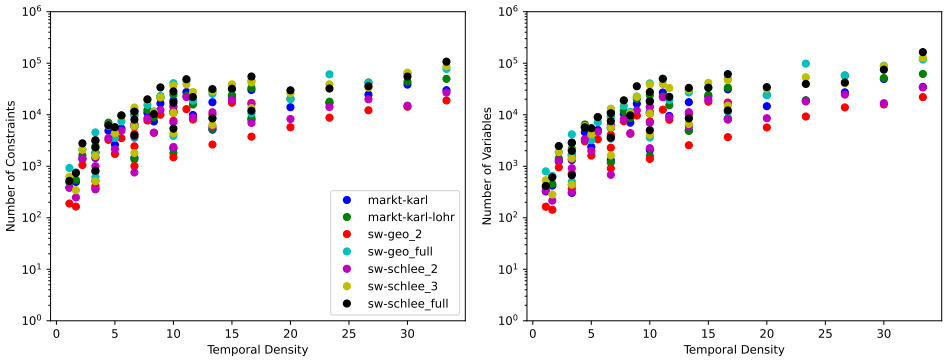
Both the absolute number of requests and the number of requests per hour have an impact on the number of admissible events in the event-based graph. Indeed, Figure 6 demonstrates that the computation time for the MILP increases with the size of the underlying event-based graph. This underlines the value of effective pre-processing approaches. Furthermore, we observe that, in our experiments, the ratio of number of arcs to number of nodes is quasi constant across instances and networks, suggesting that our pre-processing effectively prunes a large fraction of infeasible events and arcs.

3.3 Service Quality Metrics

As a large MIP gap indicates a far-from-optimal solution, we restrict our analysis to instances with ≤ 60 requests in three hours and ≤ 90 requests in six hours for this section.

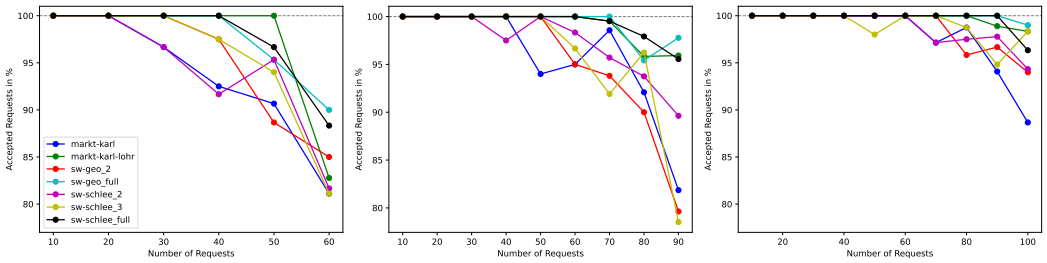
Figure 8 shows that all instances accepted over 78% of requests, with those spanning nine hours accepting over 88%. Note that it may not be feasible to accept all requests in our instances with the given number of vehicles (e.g., the optimal solution on sw-schlee_2 with 30 requests in three hours denies a single request). Instances which were solved to optimality accepted the maximum amount of passengers due to our choice of penalty W (see Lemma 1) and instances with the lowest acceptance rates correspond to those with the highest MIP gaps, compare Figure 4. Among all instances solved to optimality, 99.69% of requests were accepted on average.

As a measure of routing quality, [19] propose the metric *system efficiency*, computed as the fraction of total booked kilometers (as a straight-line path between origin and destination) to total driven distance by all vehicles. A value greater than 1 corresponds to our system saving distance compared to each passenger traveling in their own vehicle.



(a) Number of constraints. **(b)** Number of variables.

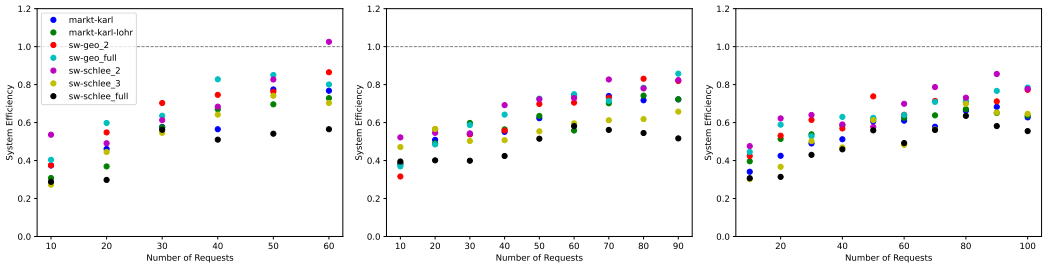
Figure 7 Number of constraints and variables in the MILP by temporal density.



(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 8 Average percentage of accepted requests per instance.

As can be seen in Figure 9, the system efficiency increases with the number of requests of the instances and temporal density, with more dense instances achieving a higher system efficiency, but only one instances surpasses a value of 1. The differences across networks, where networks with more routing freedom (e.g., the sw-geo_full) achieve higher efficiencies, suggests that the liDARPT can better utilize the network here.

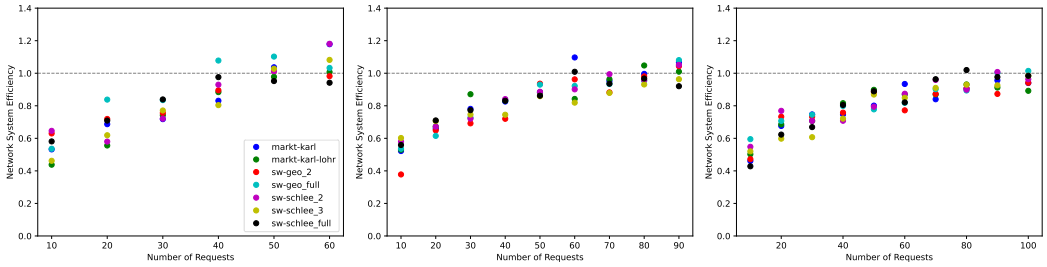


(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 9 Average system efficiency per instance.

The *system efficiency* metric may be overly harsh in evaluating liDARPT outcomes, as road network distances would, in most cases, significantly exceed the straight line distances used here for comparison. For this reason, we complement the *system efficiency* metric with the *network system efficiency*, which is defined as the ratio of $\sum_{r \in R} t_{r^*}$, the sum of the shortest network travel time of all requests, to the total distance traveled of all vehicles.

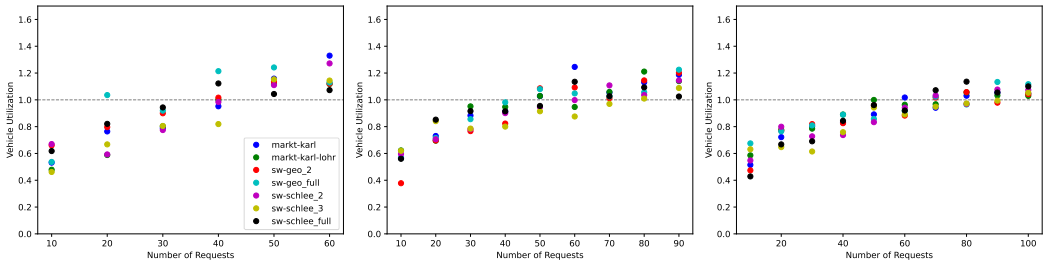
Figure 10 shows that the network system efficiency surpasses the threshold of 1 multiple times, with the lowest efficiency slightly below 0.4 and overall densely clustered results. The results show a clear trend of increasing network system efficiency with larger temporal density, where notably several instances surpass the threshold of 1, indicating that the solver is able to identify more efficient paths than those in a fixed bus network.



(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 10 Average network system efficiency per instance.

Lastly, Figure 11 considers the *vehicle utilization*[5], which increases with the number of requests, surpassing the threshold of 1 in all three time spans for higher request numbers, and reaches values up to 1.32 for larger instances. This suggests that the liDARPT is able to effectively pool passengers for medium and high densities.



(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 11 Average vehicle utilization per instance. The dashed line indicates a utilization of one.

In general, smaller instances achieve a lower utilization due to limited flexibility, resulting in longer trips to reposition the vehicles between requests. Differences in network structures may additionally play a role: the sw-schlee_3 network contains two relatively long lines without a transfer point at their end, resulting in many empty driven kilometers, e.g., after a request has been transported to the end of a line and the vehicle needs to return empty, and thus an overall lower utilization when compared to similarly sized networks. However, we cannot observe a general trend related to the networks: the vehicle utilization seems to correlate strongly with the number of requests but not with the number of lines or transfer points in the network.

[5] ratio of total passenger kilometers to the total vehicle distance traveled.

4 Conclusion

Inspired by the combination of scheduled bus services and dial-a-ride systems, we present the novel line-based dial-a-ride problem with transfers (liDARPT), wherein vehicles operate on-demand on a multi-line network, allowing passengers to transfer during their journey.

Our computational experiments with benchmark instances based on long-distance bus networks demonstrate the viability of our solution approach for the liDARPT under varying conditions, examining effects of increasing number of passengers, total time span, and number of lines in the network. As a feasibility study, our results show that the liDARPT can accept a large amount of passengers, and additionally pool these passengers in the available vehicles, achieving environmental savings by reducing the traveled distance. Hence, the model is a viable alternative to public transport, enabling a more flexible usage of existing infrastructure, especially in regions with low demand.

In the static variant studied here, our approach was able to solve instances with up to 80 requests in the given time limit of 15 minutes. Across all instances solved to optimality, 99.69% of requests were accepted, and, for increasing instance size, both the network system efficiency and the vehicle utilization surpass their threshold of 1.

To better address passenger satisfaction, future work could incorporate the generalized travel time, i.e., the sum of travel, waiting, and transfer times, in the objective function as a measure of service quality. Beyond this, further research should concentrate on the development and evaluation of solution approaches for the dynamic liDARPT, and the comparison to public transport systems operated fully on demand, on the one hand, and scheduled public transport, on the other hand. Furthermore, future work on the liDARPT may explore an extension where vehicles may be dynamically reassigned between lines, allowing a responsive (or preemptive) approach to variable demand and a varying amount of vehicles per line. As a first step, the current MILP formulation may be extended to determine the optimum number of vehicles per line for the given instance, subject to a global fleet size constraint. Building on this, a second step could incorporate vehicle transfers between lines at the transfer stations, which would require significant modifications to the current underlying event-based graph structure.

References

1 Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017. doi:10.1073/pnas.1611675114.

2 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, March 2004. doi:10.1016/j.parco.2003.12.001.

3 Jonas Barth. liDARPT. Software, swhId: swh:1:dir:f4c2255f2efac8e14d97cf8b152dd7df1a83841a (visited on 2025-08-27). URL: https://github.com/barjon0/liDARPT, doi:10.4230/artifacts.24588.

4 Jonas Barth. The line-based dial-a-ride problem with transfers. Master's thesis, Julius-Maximilians-Universität Würzburg, Germany, 2025.

5 John W. Baugh, Gopala Krishna Reddy Kakivaza, and John R. Stone. Intractability of the Dial-a-Ride Problem and a Multiobjective Solution Using Simulated Annealing. *Engineering Optimization*, 30(2):91–123, February 1998. doi:10.1080/03052159808941240.

6 Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. A Hybrid Tabu Search and Constraint Programming Algorithm for the Dynamic Dial-a-Ride Problem. *INFORMS Journal on Computing*, 24(3):343–355, 2012. doi:10.1287/IJOC.1110.0454.

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

9 Jean-François Cordeau and Gilbert Laporte. The Dial-a-Ride Problem (DARP): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2):89–101, June 2003. doi:10.1007/s10288-002-0009-8.

10 Samuel Deleplanque and Alain Quilliot. Dial-a-ride problem with time windows, transshipments, and dynamic transfer points. *IFAC Proceedings Volumes*, 46(9):1256–1261, 2013. 7th IFAC Conference on Manufacturing Modelling, Management, and Control. doi:10.3182/20130619-3-RU-3018.00435.

11 Daniela Gaul, Kathrin Klamroth, Christian Pfeiffer, Michael Stiglmayr, and Arne Schulz. A tight formulation for the dial-a-ride problem. *European Journal of Operational Research*, 321(2):363–382, 2024. doi:10.1016/j.ejor.2024.09.028.

12 Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr. Solving the Dynamic Dial-a-Ride Problem Using a Rolling-Horizon Event-Based Graph. In *DROPS-IDN/v2/document/10.4230/OASIcs.ATMOS.2021.8*. Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2021. doi:10.4230/OASIcs.ATMOS.2021.8.

13 Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr. Event-based MILP models for ridepooling applications. *European Journal of Operational Research*, 301(3):1048–1063, September 2022. doi:10.1016/j.ejor.2021.11.053.

14 Konstantinos Gkiotsalitis and A Nikolopoulou. The multi-vehicle dial-a-ride problem with interchange and perceived passenger travel times. *Transportation research part C: emerging technologies*, 156:104353, 2023. doi:10.1016/j.trc.2023.104353.

15 Timo Gschwind and Stefan Irnich. Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. *Transportation Science*, 49(2):335–354, 2015. doi:10.1287/trsc.2014.0531.

16 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.

17 Ailing Huang, Ziqi Dou, Liuzi Qi, and Lewen Wang. Flexible Route Optimization for Demand-Responsive Public Transit Service. *Journal of Transportation Engineering Part A Systems*, 146, September 2020. doi:10.1061/JTEPBS.0000448.

18 Antonio Lauerbach, Kendra Reiter, and Marie Schmidt. The complexity of counting turns in the line-based dial-a-ride problem. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 85–98. Springer, 2025. doi:10.1007/978-3-031-82697-9_7.

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

20 Renaud Masson, Fabien Lehuédé, and Olivier Péton. The dial-a-ride problem with transfers. *Computers & Operations Research*, 41:12–23, 2014. doi:10.1016/j.cor.2013.07.020.

21 Yves Molenbruch, Kris Braekers, and An Caris. Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, 259(1):295–325, December 2017. doi:10.1007/s10479-017-2525-0.

22 Sophie N. Parragh, Jean-François Cordeau, Karl F. Doerner, and Richard F. Hartl. Models and algorithms for the heterogeneous dial-a-ride problem with driver-related constraints. *OR Spectrum*, 34(3):593–633, 2012. doi:10.1007/s00291-010-0229-9.

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

24 Christian Pfeiffer and Arne Schulz. An ALNS algorithm for the static dial-a-ride problem with ride and waiting time minimization. *OR Spectrum*, 44(1):87–119, 2022. doi:10.1007/s00291-021-00656-7.

25 Line Blander Reinhardt, Tommy Clausen, and David Pisinger. Synchronized dial-a-ride transportation of disabled passengers at airports. *European Journal of Operational Research*, 225(1):106–117, February 2013. doi:10.1016/j.ejor.2012.09.008.

26 Kendra Reiter, Marie Schmidt, and Michael Stiglmayr. The line-based dial-a-ride problem. In *24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, 2024. doi:10.4230/OASIcs.ATMOS.2024.14.

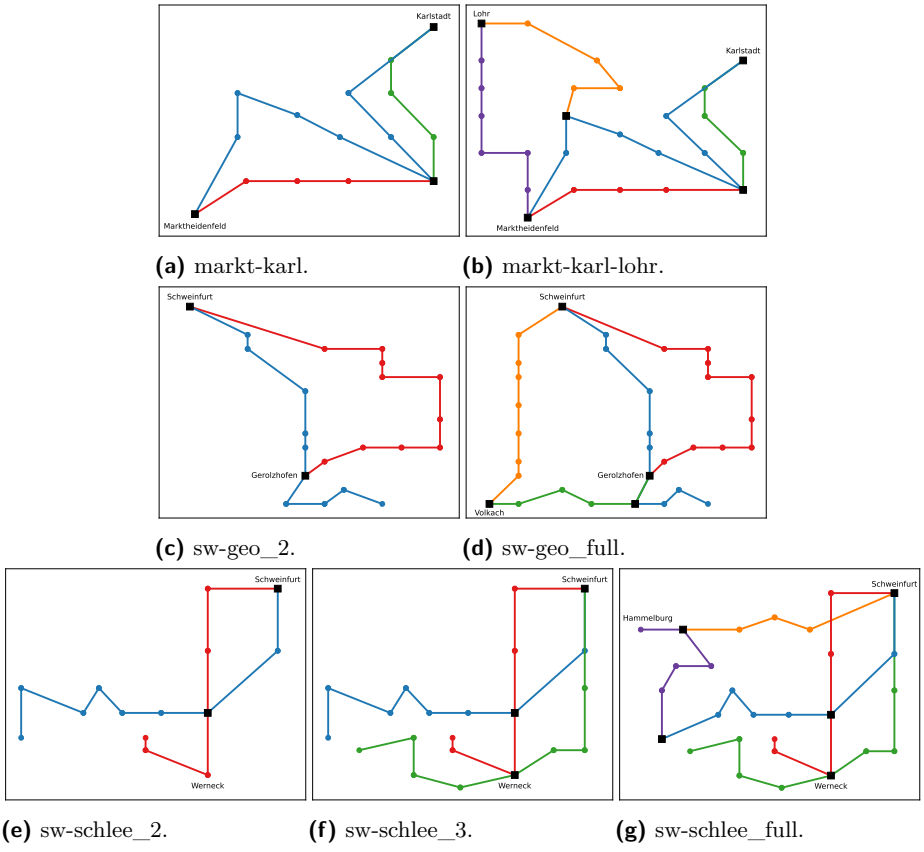
27 Yannik Rist and Michael A. Forbes. A new formulation for the dial-a-ride problem. *Transportation Science*, 55(5):1113–1135, 2021. doi:10.1287/trsc.2021.1044.

28 Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007. doi:10.1002/net.20177.

29 Stefan Ropke and David Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, 2006. doi:10.1287/trsc.1050.0135.

30 Jörn Schönberger. Scheduling constraints in dial-a-ride problems with transfers: a metaheuristic approach incorporating a cross-route scheduling procedure with postponement opportunities. *Public Transport*, 9:243–272, 2017. doi:10.1007/s12469-016-0139-6.

A Benchmark Networks



(a) markt-karl. **(b)** markt-karl-lohr.

(c) sw-geo_2. **(d)** sw-geo_full.

(e) sw-schlee_2. **(f)** sw-schlee_3. **(g)** sw-schlee_full.

Figure 12 Visualization of the underlying networks of our benchmark instances. The black square markers signify the transfer stations.

B Variable Overview

Table 2 Summary of notation.

Identifier	Definition
Parameters	
$\mathcal{S}$	set of stations
$\mathcal{S}^T$	set of transfer stations
$\mathcal{I}$	set of lines
$\mathcal{K}$	set of vehicles
$\mathcal{R}$	set of requests
λ_i	sequence of stations of a line i
$\sigma(k)$	line of bus k
$\sigma^{-1}(i)$	set of vehicles assigned to line i
c_i	capacity of vehicles on line i
s_i^{depot}	start and end depot of a line i
$t(s, s')$	(time) distance between stations s, s'
q_r	number of passengers in request r
r^+, r^-	pick-up, drop-off station for request r
$[e(r^+), l(r^+)]$	pick-up time window for request r
$[e(r^-), l(r^-)]$	drop-off time window for request r
L_r	maximum travel time of request r
b	service time
$\mathcal{P}$	set of actions
$\mathcal{P}(r)$	set of actions of request r
$\mathcal{P}(r)^+, \mathcal{P}(r)^-$	set of outbound and inbound actions of request r
$\mathcal{P}(r)_j^+$	set of outbound actions in route option ϕ_j^r of request r
$\Phi(r)$	set of route options of request r
$\Psi(r)$	set of splits of a request r
$\rho_{s,r}^{i,+}, \rho_{s,r}^{i,-}$	outbound and inbound action of request r at station s on line i
$\phi_j^r = (\rho_{r^+,r}^{i,+}, \dots, \rho_{r^-,r}^{i',-})$	route option of request r
$\psi_{s,s',r}^i = (\rho_{s,r}^{i,+}, \rho_{s',r}^{i,-})$	split of a request r on line i from s to s'
$\text{dir}(i, s, s')$	travel direction of a split $\psi_{s,s',r}^i$ on line i
$G = (\mathcal{V}, \mathcal{A})$	event-based graph
$(v_1, v_2, \dots)$	event node in $\mathcal{V}$
$\mathbf{0}_i$	empty event at depot of line i
$\delta^{\text{in}}(v), \delta^{\text{out}}(v)$	incoming and outgoing edges of node v
Model Variables	
p_r	binary variable, is 1 if request r is accepted
x_a	binary variable, is 1 if arc a is selected
y_j^r	binary variable, is 1 if route option ϕ_j^r is selected
$B_{\rho_{s,r}^{i,\pm}}$	continuous variable, end time of the action $\rho_{s,r}^{i,\pm}$

C Summary of Results

The following tables provide a summary of results, averaged over all networks, per time span. MaxOcc denotes the maximum occupancy in all vehicles.

Table 3 Averaged results over all networks.

(a) Three hour time span.

#Req	MIPGap	AccReq	SysEff	NetEff	VehUtil	MaxOcc	Avg#Transfers
10	0.00	100.00	0.37	0.55	0.57	1.71	0.74
20	0.00	100.00	0.46	0.67	0.75	3.14	0.68
30	0.00	99.05	0.60	0.77	0.85	3.14	0.61
40	1.57	97.02	0.66	0.91	1.02	4.14	0.70
50	5.57	94.38	0.74	1.02	1.14	4.05	0.68
60	18.52	84.29	0.78	1.06	1.17	4.43	0.71
70	27.90	78.23	0.81	1.09	1.22	5.05	0.65
80	40.43	71.37	0.82	1.07	1.18	4.48	0.59
90	48.95	67.41	0.81	1.08	1.19	4.48	0.58
100	73.05	58.05	0.80	1.02	1.11	4.05	0.51

(b) Six hour time span.

#Req	MIPGap	AccReq	SysEff	NetEff	VehUtil	MaxOcc	Avg#Transfers
10	0.00	100.00	0.41	0.53	0.56	1.86	0.60
20	0.00	100.00	0.51	0.67	0.75	2.43	0.69
30	0.00	100.00	0.53	0.76	0.85	2.86	0.81
40	0.00	99.64	0.56	0.80	0.90	3.29	0.79
50	0.38	99.14	0.64	0.89	1.01	3.86	0.77
60	1.29	97.86	0.67	0.94	1.05	4.14	0.74
70	1.95	97.01	0.70	0.93	1.04	4.48	0.72
80	4.76	94.46	0.72	0.98	1.10	4.67	0.74
90	13.43	88.41	0.73	1.02	1.14	4.67	0.70
100	21.19	82.57	0.75	1.00	1.12	4.24	0.64

(c) Nine hour time span.

#Req	MIPGap	AccReq	SysEff	NetEff	VehUtil	MaxOcc	Avg#Transfers
10	0.00	100.00	0.38	0.50	0.55	1.86	0.64
20	0.00	100.00	0.48	0.68	0.74	2.43	0.82
30	0.00	100.00	0.53	0.70	0.75	2.71	0.66
40	0.00	100.00	0.54	0.76	0.83	2.71	0.77
50	0.00	99.71	0.62	0.84	0.92	3.52	0.71
60	0.00	100.0	0.60	0.85	0.93	3.43	0.72
70	0.00	99.18	0.65	0.90	1.00	3.57	0.69
80	0.81	98.69	0.69	0.93	1.03	4.14	0.75
90	2.33	97.46	0.70	0.95	1.05	4.67	0.75
100	4.48	95.57	0.69	0.96	1.07	4.14	0.75