The Complexity of Counting Turns in the Line-Based Dial-a-Ride Problem

Antonio Lauerbach(✉), Kendra Reiter, and Marie Schmidt

Department of Computer Science, University of Würzburg, Würzburg, Germany
antonio.lauerbach@stud-mail.uni-wuerzburg.de,
{kendra.reiter,marie.schmidt}@uni-wuerzburg.de

Abstract. Dial-a-Ride problems have been proposed to model the challenge to consolidate passenger transportation requests with a fleet of shared vehicles. The LINE-BASED DIAL-A-RIDE PROBLEM *(*LIDARP*)* is a variant where the passengers are transported along a fixed sequence of stops, with the option of taking shortcuts. In this paper we consider the LIDARP with the objective function to maximize the number of transported requests. We investigate the complexity of two optimization problems: the LIDARP, and the problem to determine the minimum number of turns needed in an optimal LIDARP solution, called the MINTURN *problem*. Based on a number of instance parameters and characteristics, we are able to state the boundary between polynomially solvable and NP-hard instances for both problems. Furthermore, we provide parameterized algorithms that are able to solve both the LIDARP and MINTURN problem.

Keywords: Dial-a-Ride Problem · liDARP · NP-hardness · Parameterized Complexity

1 Introduction

Ridepooling, i.e., to flexibly serve passenger transportation requests with a fleet of shared vehicles, has been recognized as a promising option to replace conventional public transport, in particular in regions with low demand density. In this paper, we study the complexity of the LINE-BASED DIAL-A-RIDE PROBLEM (LIDARP), which combines the spatial aspects of a fixed sequence of stops (utilizing existing infrastructure) with the temporal flexibility of ridepooling. The goal is to reduce mobility-related emissions by efficiently pooling passengers with an improved service quality due to the fixed spatial structure. Here, we consider the objective to maximize the number of transported passengers.

Secondly, we study the complexity of determining the minimum number of turns per vehicle in an optimal LIDARP solution, which we refer to as the MINTURN problem. This problem may be relevant in practice, e.g., when turns of autonomous vehicles need to be supervised by a (remote) operator. It can also be relevant when formulating the LIDARP as a mixed-integer linear program, compare [15].

The remainder of the paper is organized as follows. In Sect. 2, we formally define the LIDARP and the MINTURN problems. We summarize known complexity results and further related research findings from prior work in Sect. 3, before we give an overview of the contribution of this paper in Sect. 4. In the following Sects. 5 and 6, we are able to characterize the boundary between polynomially solvable and NP-hard cases of LIDARP and MINTURN according to instance specifics. In Sect. 7 we provide parameterized algorithms for the LIDARP and MINTURN problem. We conclude by discussing open problems for further research in Sect. 8. The (full) proofs of statements marked with a "$\star$" are in the full version of this paper [13].

2 Problem Definition

We start by defining the LINE-BASED DIAL-A-RIDE PROBLEM (LIDARP) based on the definitions by Reiter et al. [15]: a *line*, given by a sequence H of h *stops*, is operated by k *vehicles*, each with *capacity* c, that transport some of the n *passenger requests* P.

The *(time) distance* between distinct stops $i, j \in H$ is given by $t_{i,j} \in \mathbb{N}$. The vehicles may take *shortcuts* by skipping stops, *wait* at a stop, or *turn* (i.e., change direction with respect to the sequence of stops prescribed by the line) at a stop. To turn, a vehicle needs $t_{\text{turn}} \in \mathbb{N}_0$ time (the *turn time*).

Each passenger submits a *request* $p \in P$ for transportation from an *origin* $o_p \in H$ to a *destination* $d_p \in H$ with $o_p \neq d_p$. If o_p precedes d_p in the sequence of stops, we say that the request p is *ascending*, otherwise it is *descending*. In contrast to [15], we assume that each passenger submits an individual requests, i.e., we do not allow group requests and thus the *passenger load* of each requests is 1. A request may specify a *time window* $[e_p, l_p]$, delimited by an *earliest pick-up time* e_p and a *latest drop-off time* l_p, during which it can be served. We therefore can write a request p as $([o_p, d_p], [e_p, l_p])$. Picking up or dropping off a passenger requires a *service time* of $t_{\text{s}} \in \mathbb{N}_0$ per passenger. Further, a passenger may not leave the vehicle until arriving at their destination. We make the *service promise* that the ride time of a passenger p may not exceed $\alpha \geq 1$ times the *direct time distance* t_{o_p,d_p}, that is, the *maximum ride time* of a passenger is $\alpha \cdot t_{o_p,d_p}$. The *ride time* is measured from the end of pick-up to the beginning of drop-off.

In the LIDARP, we further guarantee that if we pick up a passenger, the passenger is at all times transported towards their destination (regarding the sequence of stops). Consequently, a turn is only allowed for vehicles without passengers on board. This so-called *directionality* property [15] constitutes the main difference between the LIDARP and the 'regular' DARP.

A *tour* r consists of a sequence of *timestamped waypoints*, each waypoint being a pick-up/drop-off of a request with the timestamp corresponding to the start of the pick-up/drop-off. In order for a tour to be *feasible*, the timestamps need to adhere to the time constraints imposed by the time distances, as well as the service and turn times, the service promises, and the time windows which delimit the start of pick-ups and drop-offs. Furthermore, at no point in time may there be more than c passengers in each vehicle and each request may only be served once. If we remove the timestamps from the waypoints, we obtain the *route* underlying a tour. A route is *feasible* if it can be complemented to a feasible tour by adding timestamps.

A given tour r can be decomposed into segments, called *subtours*, with the vehicle turning precisely at the end of each subtour. Note that each subtour is, on its own, a tour, thus inheriting the feasibility definition. We analogously define *subroutes* for routes. Similar to requests, we say that a subtour/subroute is *ascending* if the vehicle drives along the sequence of stops, and *descending* if it drives the sequence in reverse. Thus, due to the directionality property, ascending requests must be served in ascending subtours and descending requests in descending subtours. We denote by $|r|$ the number of turns of a tour r. Note that this corresponds to the number of subtours of r as a vehicle always turns at the end of a subtour.

A collection of tours R is *feasible*, if each tour $r \in R$ is feasible and each request is served at most once. We analogously define *feasible collections of routes.*

A *solution* to the liDARP is a feasible collection of up to k tours. The liDARP (as we consider it here) consists of determining a solution that maximizes the number of served requests.

Given a liDARP instance, the MinTurn problem determines the minimum over the largest number of turns a vehicle has to take in an optimal liDARP solution for this instance. MinTurn thus determines $\tau := \min_{R\in\mathcal{R}^*} \max_{r\in R}|r|$ where $\mathcal{R}^*$ is the set of optimal solutions for the given liDARP instance.

Conventions. We assume that time starts at 0 and is integer. We consider all cases as special cases of the liDARP: to omit the service promise, we set $\alpha = \infty$, the service and turn times can be omitted by setting $t_{\mathrm{s}} = 0$ and $t_{\mathrm{turn}} = 0$, and the time windows can be disabled by setting $e_p = 0$ and $l_p = \infty$ for all $p \in P$. In the case without time windows, a request $p \in P$ is thus specified only by its origin and destination, i.e., $p = ([o_p, d_p])$.

We say that two requests *overlap* if they are in the same direction and the intervals between their respective origin and destination are not interior disjoint.

We assume k and c to be bounded by the number of requests n. Given a positive integer n, we use $[n]$ as shorthand for $\{1, 2, \dots, n\}$.

3 Related Work

The Dial-a-Ride Problem has been extensively studied in the literature, with a focus on modelling approaches using mixed-integer linear programs and solution strategies including both exact strategies and heuristics. The surveys by Ho et al. [8] and Vansteenwegen et al. [17] provide a comprehensive overview.

In [15], Reiter et al. introduce the *Line-Based Dial-a-Ride Problem (liDARP)*, where they aim to find a solution which maximizes a weighted sum of transported requests and saved distance (i.e., the difference between the sum of direct distances of transported passengers and the total distance driven by vehicles). The version of the LIDARP studied here is a special case of that problem, as we consider only one of the objectives. Reiter et al. [15] propose and compare three different mixed-integer linear formulations, including the *subline-based* formulation which explicitly models sequences of turns for each vehicle.

The complexity of DARP on a line with *makespan objective*, minimizing the *completion time* (the time to serve all requests), has been addressed by a number of publications in the literature. We summarize their findings in Table 1, where $o = d$ denotes the setting where all requests' origins are equal to their destinations (equivalent to the TRAVELLING SALESPERSON PROBLEM). Furthermore, some publications ([3,16]) consider *individual* service times t_s per request.

All publications listed in Table 1 fix the vehicles' starting positions, consider a *closed* setting, where the vehicles have to return to their starting position at the end of the day, and require all n requests to be served.

Table 1. Overview of known results for DARP on a line with makespan objective.

#Veh.	Cap.	$o = d$	Time Windows	Complexity	Ref.	Comment
1	1	–	–	polynomial	[14]	
1	1	–	e_p	NP-complete	[2]	Thm 7.6
1	2	–	–	NP-complete	[6]	
1	≥ 2	–	–	NP-complete	[2]	Thm 7.8
1	∞	–	–	polynomial	[14]	
1	≥ 1	✓	e_p	NP-complete	[16]	individual t_s
1	≥ 1	✓	$[e_p, l_p]$	NP-complete	[16]	
2	1	✓	$[e_p, l_p]$	NP-complete	[3]	individual t_s
≥ 1	c	✓	l_p	polynomial	[14]	

We note that de Paepe [14] further showed that the setting with an arbitrary number of vehicles of fixed capacity c, without time windows and where $o = d$, is also polynomially solvable under the objective of minimizing the sum of (weighted) completion times.

Further research has been conducted into examining the complexity of the Dial-a-Ride problem with individual loads per requests, minimizing the sum of driven distances on the half-line and line, as well as on the star, tree, circle, and $\mathbb{R}^d$ with the Euclidean metric [1,14].

Lastly, approximation algorithms for the related Vehicle Scheduling Problem on a line (L-VSP) have also been proposed. Karuno et al. [12] developed a $\frac{3}{2}$-approximation algorithm for the closed L-VSP with a single vehicle and time windows, under minimizing completion times, where the starting vertex is fixed. Allowing for arbitrary starting vertices, Gaur et al. [5] develop a $\frac{5}{3}$-approximation for the L-VSP. Under the objective of minimizing makespan, with an arbitrary starting position, Yu and Liu [18] develop a $\frac{3}{2}$-approximation for the closed, and a $\frac{5}{3}$-approximation algorithm for the open variants.

Considering the Multi-Vehicle Scheduling Problem (MVSP) on a line, Karuno and Nagamochi [11] present a 2-approximation algorithm to minimize the total completion time for a fixed number of vehicles.

4 Our Contribution

In this paper, we study the complexity of the liDARP and the novel MinTurn problem. Unlike the DARP on a line studied in the literature, we consider an *open* setting, where the vehicles do not have to return to their (arbitrary) starting positions, and maximize the number of served passengers as an objective. Note that most of our results can be transferred to the closed setting.

We consider different instance parameters and characteristics: the number of vehicles k and their capacity c, as well as the presence of time windows, shortcuts, turn times, the service promise, and the service time. Our complexity results for the MinTurn problem are summarized in Table 2 and novel results for the liDARP are given in Table 3. Note that whether there is a turn time or not appears to be irrelevant for the complexity of the problems: we have no turn time in all hardness results, while all algorithmic results hold for arbitrary turn times. We further show that it is strongly NP-hard to approximate MinTurn with a factor better than 3 (Corollary 1).

Table 2. Overview of novel results for the MinTurn problem presented here.

#Veh.	Cap.	Time Windows	Shortcuts	Service Promise	Service Time	Complexity	Ref.
≥ 1	≥ 1	–	–	✓	–	polynomial	Thm. 2
≥ 1	≥ 1	–	✓	–	✓	polynomial	Thm. 2
≥ 1	1	–	✓	✓	✓	polynomial	Thm. 2
≥ 1	≥ 2	–	–	✓	✓	strongly NP-hard	Thm. 3
≥ 1	≥ 2	–	✓	✓	–	strongly NP-hard	Thm. 4
≥ 1	≥ 1	✓	–	–	–	strongly NP-hard	Thm. 5

Table 3. Overview of novel results for the liDARP problem presented here.

#Veh.	Cap.	Time Windows	Shortcuts	Service Promise	Service Time	Complexity	Ref.
≥ 1	≥ 1	–	✓	✓	✓	polynomial	Thm. 1
≥ 1	≥ 1	✓	–	–	–	strongly NP-hard	Thm. 5

5 Polynomially Solvable Cases

In this section, we characterize the cases in which the liDARP and MinTurn problem are polynomially solvable. We begin by showing that it suffices to consider feasible routes, as we can efficiently transform them into feasible tours.

Lemma 1. (⋆). *Given a route, we can check in polynomial time whether it is feasible and, if so, complement it to a feasible tour. If there are no time windows, this can even be done in linear time. If additionally, there is no service promise, the route is feasible as long as it respects capacities.*

Without time windows it even suffices to find feasible subroutes, as any route obtained by *joining* feasible routes, i.e., concatenating the sequences of waypoints of the routes, is feasible.

Lemma 2. (⋆). *Consider a* liDARP *instance without time windows and a feasible collection R of routes. Joining all routes in R (in arbitrary order) yields a feasible route.*

We now use these lemmas, to show that the liDARP is polynomially solvable in the absence of time windows, by constructing a feasible tour that serves all requests consecutively.

Theorem 1. (⋆). *If there are no time windows, all requests can be served. A solution for the* liDARP *serving all requests can be computed in linear time.*

As we see later in Theorem 5, the liDARP is NP-hard as soon as we have time windows. We therefore now focus on the MinTurn problem. We begin by showing that if we have already determined the (number of) subroutes needed to serve all requests, we can efficiently compute τ for the MinTurn problem.

Lemma 3. *Consider an instance of the* MinTurn *problem without time windows. Let a (b) be the smallest number of feasible ascending (descending) subroutes needed to serve all ascending (descending) requests. Assume w.l.o.g. that $a \geq b$. Then, $\tau = \max\{\lceil \frac{a+b}{k} \rceil, 2 \cdot \lceil \frac{a}{k} \rceil - 1\}$.*

Proof. We observe that we can create a feasible collection of k routes serving all requests by alternatingly joining ascending and descending subroutes into routes of length $2\lceil \frac{a}{k} \rceil$, using each subroute once and adding *artificial subroutes*, that do not serve requests, in case there not enough subroutes. The resulting routes are feasible according to Lemma 2, as all subroutes, including the artificial subroutes, are feasible.

Furthermore, we can prove two lower bounds on the number of turns per vehicle: First, by the pigeonhole principle, there has to be a route consisting of at least $\lceil \frac{a+b}{k} \rceil$ subroutes. Second, as there must be a route containing at least $\lceil \frac{a}{k} \rceil$ ascending subroutes, and ascending and descending subroutes alternate, this route contains at least $2\lceil \frac{a}{k} \rceil - 1$ subroutes. The lower bound is thus $\max\{\lceil \frac{a+b}{k} \rceil, 2\lceil \frac{a}{k} \rceil - 1\}$.

If $\lceil \frac{a+b}{k} \rceil = 2\lceil \frac{a}{k} \rceil$, the upper and lower bound coincide. Otherwise, it holds that $\lceil \frac{a+b}{k} \rceil \leq 2 \cdot \lceil \frac{a}{k} \rceil - 1$. In this case, we add d artificial descending subroutes such that $\frac{a+b+d}{k} = 2\lceil \frac{a}{k} \rceil - 1$. We then combine the subroutes into a feasible collection of k routes serving all requests, each with $2\lceil \frac{a}{k} \rceil - 1$ turns. For this, we first add $\lceil \frac{a}{k} \rceil - 1$ ascending subroutes as well as $\lceil \frac{a}{k} \rceil - 1$ descending subroutes to each route. This leaves exactly k subroutes unassigned, as $a + b + d = \big(2\lceil \frac{a}{k} \rceil - 1\big) \cdot k = 2k(\lceil \frac{a}{k} \rceil - 1) + k$. We can thus assign each of these k subroutes to a separate route. Thus, each route is assigned $2\lceil \frac{a}{k} \rceil - 1$ subroutes, with a route either containing $\lceil \frac{a}{k} \rceil$ ascending or $\lceil \frac{a}{k} \rceil$ descending subroutes. By alternating the ascending and descending subroutes in a route, we obtain routes with $2\lceil \frac{a}{k} \rceil - 1$ turns. According to Lemma 2, these routes form a feasible collection. □

To solve the MinTurn problem in the absence of time windows, we thus need to determine the minimum number of feasible ascending and descending subroutes needed to serve all requests. We now show that, if the feasibility of subroutes is determined by the capacity, this can be done in polynomial time.

Lemma 4. *Consider a* MinTurn *instance where subroutes are already feasible if they respect the capacity. Let χ be the maximum number of pairwise overlapping ascending (descending) requests. The minimum number of feasible ascending (descending) subroutes needed to serve all ascending (descending) requests is $\lceil \chi/c \rceil$. Determining χ is possible in polynomial time.*

Proof. The assignment of ascending (descending) requests to seats in subroutes corresponds to coloring the conflict graph of the requests, as no two overlapping requests may occupy the same seat. As the conflict graph is an interval graph, the chromatic number χ corresponds to the maximum number of pairwise overlapping requests [9] and can be determined in polynomial time. Thus, if χ seats are needed, $\lceil \chi/c \rceil$ subroutes are necessary to serve all requests. □

These lemmas imply that a MinTurn instance without time windows is solvable in polynomial time if the feasibility of subroutes is determined solely by the capacity constraints. We use this insight to characterize the cases in which the MinTurn problem is polynomially solvable.

Theorem 2. ($\star$). *Consider an instance of* MinTurn. *Let a (b) be the maximum number of pairwise overlapping ascending (descending) requests and assume w.l.o.g. $a \geq b$. In the following cases of the* MinTurn *problem, we have $\tau = \max\{\lceil \frac{a+b}{k} \rceil, 2 \cdot \lceil \frac{a}{k} \rceil - 1\}$, which can be determined in polynomial time:*

1. *without time windows and without service promise*
2. *without time windows, without shortcuts, and without service times*
3. *without time windows and with a capacity of* 1

6 Hardness Results

In this section, we show that all remaining liDARP and MinTurn cases are strongly NP-hard, using reductions from 3-Partition, which is well known to be strongly NP-hard, see [3].

Definition 1. (3-Partition [3]). *Given a finite set S of $n = 3m$ positive integers as well as a bound $T \in \mathbb{N}$ such that $\sum_{s \in S} s = mT$ and $T/4 < s < T/2$ for all $s \in S$, is there a partition of S into m disjoint sets $S_1, \ldots, S_m$ such that $\sum_{s \in S_j} s = T$ for all $j \in [m]$?*

We begin by showing hardness of MinTurn even in the absence of time windows and shortcuts.

Theorem 3. ($\star$). *The problem* MinTurn *is strongly NP-hard for all $k \geq 1$ and $c \geq 2$ even without time windows, shortcuts, and turn time.*

Proof. We begin by showing the reduction from 3-Partition for $k = 1$ and $c = 2$. Let (S, m, T) be an instance of 3-Partition and $S = \{s_1, \ldots, s_n\}$.

We create an instance of MinTurn such that the ascending subroutes in an optimal solution of the liDARP with minimum turns per route correspond to a 3-partition of S. An example of such an instance can be seen in Fig. 1.

We begin by having stops $H = \langle 1, \ldots, 4 + 4Tn \rangle$ with unit distance between neighboring stops. We create four types of requests: for each $i \in [n]$ we create s_i *value requests* $P_{\mathrm{V}}^i = \{([4 + T(i-1) + (j-1), 4 + T(i-1) + j]) \mid j \in [s_i]\}$ and m *plug requests* $P_{\mathrm{P}}^i = P_{\mathrm{LP}}^i \cup \{p_{\mathrm{P}}^i\}$. The plug requests consist of $m - 1$ *long plug requests* P_{LP}^i, each being $([4 + T(i-1), 4 + Ti])$, and one *short plug request* $p_{\mathrm{P}}^i = ([4 + T(i-1) + s_i, 4 + Ti])$. We also create m *promise requests* P_{SP}, each being $([1, 4 + 4Tn])$, which are used in combination with the service promise to ensure that the number of value requests in a subroute does not exceed T. Lastly, we create m *filter requests* P_{F}, each being $([2, 3])$, which are used to ensure that each subroute contains exactly one promise request. We set the service time $t_{\mathrm{s}} = 1$ and the service promise $\alpha = 1 + b/a$ with $a = 3 + 4Tn$ and $b = 2(1 + T + n)$. Note that $b < a$ and thus $\alpha < 2$.

We now show that S has a 3-partition if and only if $\tau = 2m - 1$ for the constructed MinTurn instance.

From Theorem 1, we know that in an optimal liDARP solution to the constructed instance all requests are served. As we see in Observation 1, each filter request has to be served by a different subroute, requiring at least m ascending subroutes to serve all requests. We thus need at least $2m - 1$ turns to serve all requests. Assume that we have an optimal solution to the liDARP that uses $2m - 1$ turns. Its route must thus contain m ascending subroutes.

Observation 1: Each ascending subroute serves exactly one filter and one promise request. Indeed, as the service promise α is less than 2 and the service time t_{s} is 1, the maximum ride time of a filter request is less than 2, due to its direct time distance being 1, and would thus be exceeded if another filter request is served by the same subroute. Since the m ascending subroutes serve all requests, each ascending subroute must thus contain exactly one filter request. It follows that each ascending subroute must also contain exactly one promise request, since promise and filter request overlap.

Observation 2: Due to the service promise, the maximum ride time of a promise request is at most b more than the direct time distance. As all other requests lie between the origin and destination of a promise request and the

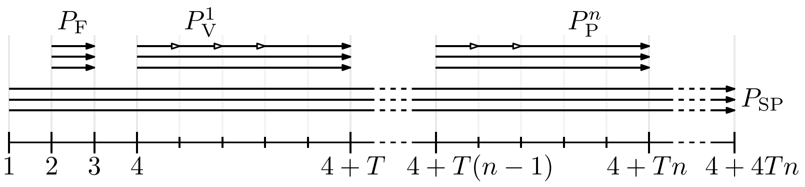

Fig. 1. A MinTurn instance constructed from a 3-Partition instance with $m = 3$ and $T = 5$, as well as $s_1 = 3$ and $s_n = 2$. The arrows represent the requests, with the white tipped arrows being value requests.

service time is 1, at most $1 + T + n$ requests can be transported in a subroute besides a promise request. According to Observation 1, one of these requests must be a filter request.

Observation 3: Each subroute also has to serve exactly one plug request for each $i \in [n]$, as there are m such requests and they overlap each other as well as the promise requests.

Observation 4: Combining Observations 2 and 3, we conclude that each subroute may serve up to T value requests. As the total number of value requests is mT, this means that each subroute transports exactly T such requests.

Observation 5: All requests in P_{V}^i must be served by the same subroute, the one that serves the short plug request p_{P}^i, as all the other subroutes contain long plug requests that overlap all requests in P_{V}^i. That is, for each $i \in [n]$ we have exactly one subroute that serves the s_i value requests P_{V}^i. In combination with Observation 4 we conclude that, for each subroute r_j, we have an index set I_j, such that all value requests in $\bigcup_{i \in I_j} P_{\mathrm{V}}^i$ are served by the subroute r_j and $\sum_{i \in I_j} s_i = T$. Setting $S_j := \{s_i \mid i \in I_j\}$, we thus obtain a valid 3-partition of S.

Conversely, if there exists a 3-partition $S_1, \ldots, S_m$ of S, we create m ascending subroutes $r_1, \ldots, r_m$ and assign for each $s_i \in S_j$ for $j \in [m]$ the value requests in P_{V}^i as well as the short plug request in p_{P}^i to the subroute r_j. To each of the remaining ascending subroutes, we assign one of the long plug requests from P_{LP}^i. Furthermore, we assign one filter and one promise request to each ascending subroute. In this way, all requests are assigned to a subroute. Additionally, the capacities are respected, as no more than 2 requests pairwise overlap in a subroute. By adding $m - 1$ artificial descending subroutes, we connect the ascending subroutes and obtain a route with $2m - 1$ turns. For this route to be feasible it remains to show, according to Lemma 2, that the subroutes are feasible. The only requests which are not transported directly are the promise requests. By construction, each subroute serves, besides the promise request, one filter, n plug, and T value requests. Therefore, the delay in the ride time of a promise request is $2(1 + T + n)$, which is precisely the allowed delay by the service promise, and the subroutes are thus feasible.

For higher capacities and more vehicles we duplicate the promise requests, such that they fill the added seats. Apart from a small adjustment of the service promise and subsequently the direct time distance of promise requests, the construction and correctness are analogous.

Constructing these instances takes pseudo-polynomial time. As 3-Partition is strongly NP-hard, it follows that the MinTurn problem is strongly NP-hard. □

The presented reduction can be adapted to show strong NP-hardness for the case without service times but instead with shortcuts, by encoding the values $s \in S$ into detours of the line that can be shortcut by a subroute if it does not serve the corresponding requests.

Theorem 4. (⋆). *The problem* MinTurn *is strongly NP-hard for all $k \geq 1$ and $c \geq 2$ even without time windows, service times and turn times.*

Proof. We begin by proving hardness for $k = 1$ and $c = 2$ before extending it to higher values. Let (S, m, T) be an instance of 3-Partition and $S = \{s_1, \dots, s_n\}$.

As we use shortcuts, we start by describing the layout of the line, which can be seen schematically in Fig. 2.

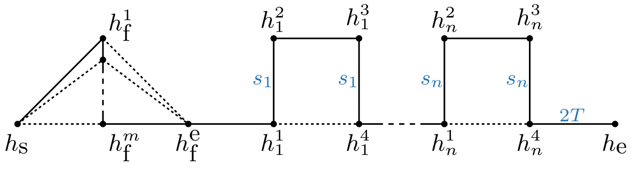

Fig. 2. The layout of the stops constructed for a 3-Partition instance. The line is the continuous path, while shortcuts are represented by (dotted) lines. The distances between neighboring stops are given by (blue) labels, with the exception of distances of 1, which are omitted.

The line starts at stop h_{s} then contains (in the order in which they are listed here) a sequence of stops h_{f}^j for $j \in [m]$, a stop $h_{\mathrm{f}}^{\mathrm{e}}$, for each $s_i \in S$ a sequence of stops $h_i^1, h_i^2, h_i^3, h_i^4$, and the final stop h_{e}. For each $i \in [n]$, the distance between h_i^1 and h_i^2 as well as between h_i^3 and h_i^4 is s_i. The distance between h_n^4 and h_{e} is $2T$. For all other pairs of subsequent stops, the distances are 1. We have a number of shortcuts: the direct distance from h_{s} to h_{f}^j and from h_{f}^j to $h_{\mathrm{f}}^{\mathrm{e}}$ is 1 for all $j \in [m]$. Furthermore, the direct distance between h_i^1 and h_i^4 is 1 for all $i \in [n]$. The m *promise requests* P_{SP} originate at h_{s} and end at h_{e}. Each of the m *filter requests* $p_{\mathrm{F}}^j \in P_{\mathrm{F}}$ for $j \in [m]$ starts at stop h_{f}^j and goes to $h_{\mathrm{f}}^{\mathrm{e}}$. For each $i \in [n]$, a *value request* p_i is added that originates at h_i^2 and goes to h_i^3. The service promise is set to $1 + b/a < 2$ with $a = 2 + 2n + 2T$ and $b = 2T$.

It then holds that S has a 3-partition if and only if $\tau = 2m - 1$ for the constructed MinTurn instance. The proof is similar to the proof of Theorem 3.

Differing from the reduction in Theorem 3, we can construct this instance in polynomial time, as our number of stops does not depend on T. It follows that the MinTurn problem is strongly NP-hard. □

By adapting the reductions to use m vehicles instead of one, we can show that the MinTurn problem is strongly NP-hard to approximate with a factor better than 3. This leads to the following corollary.

Corollary 1. ($\star$). *The problem* MinTurn *is strongly NP-hard to approximate with a factor better than* 3 *for all* $c \geq 2$ *in the following cases:*

1. *without time windows, shortcuts, and turn times, and*
2. *without time windows, service times, and turn times.*

We now show that as soon as we consider time windows, both the liDARP and MinTurn problem become NP-hard for arbitrary values of k and c.

Theorem 5. ($\star$). *Both* liDARP *and* MinTurn *are strongly NP-hard for all* $k \geq 1$ *and* $c \geq 1$ *even without shortcuts, service promise, service times, and turn times.*

Proof. (sketch). Again, we use 3-Partition for the reduction. The main idea of the proof for $k = 1$ and $c = 1$ is to translate the values $s \in S$ into value requests that need time $2s$ to be served. We then use *separator* requests to create m time intervals of length $2T$ during which the value requests must be served if all requests are served. Thus, the assignment of value requests to time intervals corresponds to a 3-partition of S. □

7 Parameterized Algorithms

Seeing as the liDARP and MinTurn problem are NP-hard, we now provide parameterized algorithms for both. Recall that an algorithm is *fixed-parameter tractable* (FPT) w.r.t. a parameter k if its runtime is $f(k) \cdot n^{O(1)}$ for some computable function f, where n is the size of the input. An algorithm is *slice-wise polynomial* (XP) w.r.t. a parameter k if its runtime is in $O(n^{f(k)})$ for some computable function f. When analyzing the runtime, we use the O^* notation, which suppresses polynomial factors in the input size, i.e., a function $g(n)$ is in $O^*(f(n))$ if there is a polynomial p such that $g(n) \in O(f(n) \cdot p(n))$.

As we have shown the liDARP and MinTurn problem to be NP-hard for constant k and c, we have to consider more parameters to obtain parameterized algorithms. We therefore use the number of stops h as well as the maximum time $t := \max_{p \in P} l_p + 1$.

Theorem 6. ($\star$). *There exists an FPT-algorithm for* MinTurn *as well as* liDARP *parameterized by* k, c, h *and* t, *with a runtime in* $O^*((h^2 \cdot t^3 \cdot c \cdot k)^{2 \cdot t \cdot c \cdot k})$.

Proof. (sketch). To prove this result, we enumerate all routes that could be part of a liDARP solution. As we can bound the number of requests that a single vehicle can serve in time t by $t \cdot c$, we can enumerate all feasible routes in time $O^*(n^{2 \cdot t \cdot c})$, using the *event-based graph* [4,15]. For all collections of up to k routes, we find the one which serves the most requests and minimizes the maximum turns per route. This results in a runtime of $O^*(n^{2 \cdot t \cdot c \cdot k})$.

To obtain an FPT-runtime, we slightly modify the algorithm such that it reduces inputs to a predetermined size, using the observation that there are at most $h^2 \cdot t^2$ distinct requests which can be served at most $t \cdot c \cdot k$ times each. □

Note that the FPT-algorithm can be adapted easily to work for several other objectives and restrictions, such as maximizing the weighted sum of saved distance and transported requests, as used in [15], or minimizing the makespan while serving all requests, as studied in most complexity papers on DARP on a line, see Table 1.

However, the case without time windows poses some difficulties. In this paper we treat this as a special case of the general case by setting $l_p = \infty$, which implies $t = \infty$. Determining a better bound for l_p is of no use, as it would depend on n. Thus, we now devise an XP-algorithm for the case without time windows whose running time is parameterized by c and h.

Theorem 7. (⋆). *There is an XP-algorithm for the* MinTurn *problem without time windows, parameterized by c and h, with runtime $O^*(n^{h^2} \cdot h^{4 \cdot c \cdot h})$.*

Proof. (sketch). We interpret finding the minimum number of feasible subroutes needed to serve all ascending (descending) requests as a Multiset Multicover problem and apply an algorithm proposed by Hua et al. [10] to solve it. Applying Lemma 3, we obtain the value of τ for MinTurn in time $O^*(n^{h^2} \cdot h^{4 \cdot c \cdot h})$. □

8 Conclusion

We introduce the MinTurn problem and characterize the boundary between polynomial solvability and NP-hardness for the MinTurn and liDARP problem according to instance specifics, including time windows, shortcuts, service promise, and service times. We also show that the MinTurn problem is strongly NP-hard to approximate with factor better than 3. We then provide an FPT-algorithm for the MinTurn and liDARP problem, parameterized by k, c, h and t, which also works for other objectives and restrictions. Finally, we present an XP-algorithm for the MinTurn problem without time windows parameterized by c and h.

An interesting topic for further research would be to analyze the liDARP and MinTurn problem for other objectives and restrictions, such as minimizing the sum of turns (over all vehicles), or allowing an unlimited number of vehicles. It also remains open, whether the parameterized algorithms can be improved, such as if there is an FPT-algorithm parameterized by c and h. For practical applications, such as the subline-based formulation [15], it may be interesting to develop heuristics and approximation algorithms for the MinTurn problem.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Archetti, C., Feillet, D., Gendreau, M., Grazia Speranza, M.: Complexity of the VRP and SDVRP. Trans. Res. Part C: Emerg. Technol. **19**(5), 741–750 (2011). https://doi.org/10.1016/j.trc.2009.12.006
2. Bjelde, A., et al.: Tight bounds for online TSP on the line. ACM Trans. Algorithms **17**(1), 3:1–3:58 (2021). https://doi.org/10.1145/3422362
3. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman (1979)
4. Gaul, D., Klamroth, K., Stiglmayr, M.: Event-based MILP models for ridepooling applications. Eur. J. Oper. Res. **301**(3), 1048–1063 (2022). https://doi.org/10.1016/J.EJOR.2021.11.053
5. Gaur, D.R., Gupta, A., Krishnamurti, R.: A 53-approximation algorithm for scheduling vehicles on a path with release and handling times. Inf. Process. Lett. **86**(2), 87–91 (2003). https://doi.org/10.1016/S0020-0190(02)00474-X
6. Guan, D.J.: Routing a vehicle of capacity greater than one. Discret. Appl. Math. **81**(1), 41–57 (1998). https://doi.org/10.1016/S0166-218X(97)00074-7
7. Haugland, D., Ho, S.C.: Feasibility testing for dial-a-ride problems. In: Chen, B. (ed.) Algorithmic Aspects in Information and Management, pp. 170–179. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14355-7_18
8. Ho, S.C., Szeto, W., Kuo, Y.H., Leung, J.M., Petering, M., Tou, T.W.: A survey of dial-a-ride problems: literature review and recent developments. Trans. Res. Part B: Methodological **111**, 395–421 (2018). https://doi.org/10.1016/j.trb.2018.02.001
9. Hougardy, S.: Classes of perfect graphs. Discrete Math. **306**(19), 2529–2571 (2006). https://doi.org/10.1016/j.disc.2006.05.021. creation and Recreation: A Tribute to the Memory of Claude Berge
10. Hua, Q., Wang, Y., Yu, D., Lau, F.C.M.: Dynamic programming based algorithms for set multicover and multiset multicover problems. Theor. Comput. Sci. **411**(26–28), 2467–2474 (2010). https://doi.org/10.1016/J.TCS.2010.02.016
11. Karuno, Y., Nagamochi, H.: A 2-approximation algorithm for the multi-vehicle scheduling problem on a path with release and handling times. In: auf der Heide, F.M. (ed.) Algorithms — ESA 2001, pp. 218–229. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). https://doi.org/10.1016/S0166-218X(02)00596-6
12. Karuno, Y., Nagamochi, H., Ibaraki, T.: Better approximation ratios for the single-vehicle scheduling problems on line-shaped networks. Networks **39**(4), 203–209 (2002). https://doi.org/10.1002/net.10028
13. Lauerbach, A., Reiter, K., Schmidt, M.: The complexity of counting turns in the line-based dial-a-ride problem (2024). https://arxiv.org/abs/2409.15192
14. de Paepe, W., Lenstra, J.K., Sgall, J., Sitters, R.A., Stougie, L.: Computer-aided complexity classification of dial-a-ride problems. INFORMS J. Comput. **16**(2), 120–132 (2004). https://doi.org/10.1287/IJOC.1030.0052
15. Reiter, K., Schmidt, M., Stiglmayr, M.: The line-based dial-a-ride problem. In: Bouman, P.C., Kontogiannis, S.C. (eds.) 24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024), pp. 14:1 – 14:20. Open Access Series in Informatics (OASIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/OASIcs.ATMOS.2024.14
16. Tsitsiklis, J.N.: Special cases of traveling salesman and repairman problems with time windows. Networks **22**(3), 263–282 (1992). https://doi.org/10.1002/net.3230220305

17. Vansteenwegen, P., Melis, L., Aktaş, D., Montenegro, B.D.G., Sartori Vieira, F., Sörensen, K.: A survey on demand-responsive public bus systems. Trans. Res. Part C: Emerg. Technol. **137**, 103573 (2022). https://doi.org/10.1016/j.trc.2022.103573
18. Yu, W., Liu, Z.: Single-vehicle scheduling problems with release and service times on a line. Networks **57**(2), 128–134 (2011). https://doi.org/10.1002/net.20393