Ride Sharing with a Vehicle of Unlimited Capacity*

Angelo Fanelli[1] and Gianluigi Greco[2]

1 **CNRS (UMR-6211), France.**
angelo.fanelli@unicaen.fr
2 **Department of Mathematics and Computer Science, University of Calabria, Italy.**
ggreco@mat.unical.it

Abstract

A ride sharing problem is considered where we are given a graph, whose edges are equipped with a travel cost, plus a set of objects, each associated with a transportation request given by a pair of origin and destination nodes. A vehicle travels through the graph, carrying each object from its origin to its destination without any bound on the number of objects that can be simultaneously transported. The vehicle starts and terminates its ride at given nodes, and the goal is to compute a minimum-cost ride satisfying all requests. This ride sharing problem is shown to be tractable on paths by designing a $O(h \log h + n)$ algorithm, with h being the number of distinct requests and with n being the number of nodes in the path. The algorithm is then used as a subroutine to efficiently solve instances defined over cycles, hence covering all graphs with maximum degree 2. This traces the frontier of tractability, since **NP**-hard instances are exhibited over trees whose maximum degree is 3.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems.

Keywords and phrases Vehicle Routing, Ride Sharing, Pick up and Delivery Problem.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.36

1 Introduction

Ride Sharing. Vehicle routing problems have been drawn to the attention of the research community in the late 50's [8]. Since then, they have attracted much attention in the literature due to their pervasive presence in real-world application scenarios, till becoming nowadays one of the most studied topics in the field of operation research and combinatorial optimization (see, e.g., [24, 29, 10] and the references therein).

Within the broad family of vehicle routing problems, a noticeable class is constituted by the pickup and delivery problems, where a given set of objects, such as passengers or goods, have to be picked at certain nodes of a transportation network and delivered at certain destinations [11]. Pickup and delivery problems can be divided in two main groups [27]. The first group refers to situations where we have a single type of object to be transported, so that pickup and delivery locations are unpaired (see, e.g., [21]). The second group deals, instead, with problems where each transportation request is associated with a specific origin and a specific destination, hence resulting in paired pickup and delivery points (see, e.g., [22, 9]).

* This work was partially supported by the project ANR-14-CE24-0007-01 *"CoCoRICo-CoDec"*. G. Greco's work was also supported by a Kurt Gödel Research Fellowship, awarded by the Kurt Gödel Society. We thank Jérôme Lang, from Université Paris-Dauphine, for introducing the subject to us.

In the paper, we focus on problems of the latter kind, and we deal with the most basic setting where *one vehicle* is available only. The vehicle is initially located at some given source node and it must reach a given destination node by means of a *feasible* ride, that is, of a ride satisfying all requests. The edges of the network are equipped with weights, and the goal is to compute an *optimal* ride, that is, a feasible ride minimizing the sum of the weights of the edges traversed by the vehicle.

Vehicles of Limited Capacity. Ride sharing with one vehicle has attracted much research in the literature and most of the foundational results in the area of vehicle routing precisely refer to this setting. In fact, earlier works have mainly focused on the case where the capacity of the vehicle is bounded by some given constant. In particular, based on whether or not we allow objects to be temporarily unloaded at some vertex of the transportation network, two versions of ride sharing problems emerge: *preemptive* (where drops are allowed) and *non-preemptive* (where drops are not allowed). An orthogonal classification comes, moreover, from the capacity c of the given vehicle. The setting with *unit* capacity ($c = 1$) has received much attention in the literature, where it often comes in the form of a *stacker crane problem* (see [15, 28] and the references therein). A natural generalization is then when the vehicle can carry more than one object at time, that is, when c is any given natural number possibly larger than 1.

Given these two orthogonal dimensions, a total of four different configurations can be studied (cf. [19]). In all the possible configurations, vehicle routing is known to be **NP**-hard [15, 16] when the underlying transportation network is an arbitrary graph. In fact, motivated by applications in a wide range of real-world scenarios, complexity and algorithms for ride sharing problems have been studied for networks with specific topologies, such as path, cycles, and trees. Consider first the unit capacity setting. In this case, ride sharing is known to be polynomial time solvable on both paths [2] and cycles [13], no matter of whether drops are allowed. Moving to trees, instead, the preemptive case remains efficiently solvable [14], while the non-preemptive case becomes **NP**-hard [12]. Consider now the case where $c \geq 1$ holds. Clearly enough, the intractability result over trees established for $c = 1$ still holds in this more general setting. In fact, in this setting, ride sharing appears to be intrinsically more complex. Indeed, it has been shown that the non-preemptive version of the problem is **NP**-hard on all the considered network topologies and that the preemptive version is **NP**-hard even on trees [18]. Good news comes instead when the problem is restricted over paths and cycles in the preemptive case. Indeed, the problem has been shown to be feasible in polynomial time on paths [19]. Moreover, the algorithm by [19] is also applicable to cycles, under the constraint that, for each object, the direction of the transportation (either clockwise, or anticlockwise) is a-priori given. More efficient algorithms are know for paths when the ride starts from one endpoint [18, 23].

Vehicles of Unlimited Capacity. There are application scenarios where the capacity of the vehicle can be better thought as being *unlimited*, as it happens, for instance, when we are transporting intangible objects, such as messages. More generally, we might know beforehand that the number of objects to be transported is less than the capacity of the vehicle; and, accordingly, we would like to use solution algorithms that are more efficient than those proposed in the literature and designed in a way that this knowledge is not suitably taken into account. In fact, the **NP**-hardness results discussed above exploit a given constant bound on the capacity and, hence, they do not immediately apply to the unbounded setting. However, specific reductions have been exhibited showing the **NP**-hardness on general graphs

(cf. [30, 3]). Moreover, heuristic methods (see, e.g., [17, 25]) and approximation algorithms (see, e.g., [1, 20]) have been defined, too. On the other hand, a number of tractability results for vehicles with unlimited capacity transporting objects of the same type can be inherited even in the paired context we are considering. Indeed, for cases where such identical objects are initially stored at the same node (or, equivalently, have to be transported to the same destination) [3, 4, 6, 5], efficient algorithms have been designed for transportation networks that are trees and cycles [30]. Moreover, the algorithm for paths proposed by [19] can be still applied over the unlimited capacity scenario. But, it was not explored so far whether better performances can be obtained by means of algorithms specifically designed for vehicles with unlimited capacity.

Contributions. The goal of the paper is to address the above research question, and to study complexity and algorithmic issues arising with ride sharing problems in presence of one vehicle of unlimited capacity. The analysis has been conducted by considering different kinds of undirected graph topologies, which have been classified on the basis of the degree of their nodes. Let n be the number of nodes in the underlying graph, let q be the number of requests (hence, of objects to be transported), and let h denote the number of distinct requests (so, $h \leq q$ and $h \leq n^2$). Then, our results can be summarized as follows:

- In Section 3, we show that optimal rides can be computed in polynomial time over graphs that are *paths*. In particular, an algorithm is exhibited to compute an optimal ride in $O(h \log h + n)$. This improves the $O(qn + n^2)$ bound that we obtain with the state-of-the-art algorithm by Guan and Zhu [19] for vehicles with limited capacity, by naïvely setting the limit to k.
- The design and the analysis of the above algorithm is the main technical achievement of the paper. By using the algorithm as a basic subroutine, we are then able to show in Section 4 that optimal rides can be computed in polynomial time over *cycles* too, formally in $O(m^2 \cdot (h \log h + n))$, with m being the number of distinct nodes that are endpoints of some request, so that $m \leq 2h$ and $m \leq n$. The result has no counterpart in the limited capacity setting, since differently from [19], we do not require that the direction of the transportation of the objects is fixed beforehand.
- Path and cycles completely cover all graphs whose maximum degree is 2. In fact, this value precisely traces the frontier of tractability for the ride sharing problem we have considered, as **NP**-hard instances are exhibited over graphs whose maximum degree is 3 and which are moreover trees.

2 Ride Sharing Scenarios

Let $G = (V, E, w)$ be an undirected weighted graph, where V is a set of nodes and E is a set of edges. Each edge $e \in E$ is a set $e \subseteq V$ with $|e| = 2$, and it is equipped with a cost $w(e) \in \mathbb{Q}^+$. A *ride* π in G is a sequence of nodes $\pi_1, \ldots, \pi_k$ such that $\pi_i \in V$ is the node reached at the *time step* i and $\{\pi_i, \pi_{i+1}\} \in E$, for each i with $1 \leq i \leq k-1$. The time step $k > 0$ is called the *length* of π, hereinafter denoted by $len(\pi)$. The value $\sum_{i=1}^{k-1} w(\{\pi_i, \pi_{i+1}\})$ is the *cost* of π (w.r.t. w) and is denoted by $w(\pi)$. Moreover, $nodes(\pi)$ denotes the set of all nodes $v \in V$ occurring in π.

A *request* on $G = (V, E, w)$ is a pair (s, t) such that $\{s, t\} \subseteq V$. Note that s and t are not necessarily distinct, and they are called the starting and terminating nodes, respectively, of the request. We say that a ride π in G *satisfies* the request (s, t) if there are two time steps i and i' such that $1 \leq i \leq i' \leq len(\pi)$, $\pi_i = s$ and $\pi_{i'} = t$. If $\mathcal{C}$ is a set of requests on G, then

$V_{\mathcal{C}}$ is the set of all starting and terminating nodes occurring in it. A *ride-sharing* scenario consists of a tuple $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$, where $G = (V, E, w)$ is an undirected weighted graph, (s_0, t_0) is a request on G and $\mathcal{C}$ is a non-empty set of requests.

A ride $\pi = \pi_1, \ldots, \pi_k$ in G is *feasible* for $\mathcal{R}$ if $\pi_1 = s_0$, $\pi_k = t_0$, and π satisfies each request in $\mathcal{C}$. The set of all feasible rides for $\mathcal{R}$ is denoted by *feasible*$(\mathcal{R})$. A feasible ride π is *optimal* if $w(\pi') \geq w(\pi)$, for each feasible ride π'. The set of all optimal rides for $\mathcal{R}$ is denoted by *opt*$(\mathcal{R})$.

Let $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$ be a ride-sharing scenario, and let π be a ride in G. Let i and i' be two time steps such that $1 \leq i \leq i' \leq len(\pi)$. Then, we denote by $\pi[i, i']$ the ride $\pi_i, \ldots, \pi_{i'}$ obtained as the sequence of the nodes occurring in π from time step i to time step i'. If π and π' are two rides on G, then we write $\pi' \preceq \pi$ if either $\pi' = \pi$ or, recursively, if there are two time steps i and i' such that $1 \leq i < i' \leq len(\pi)$, $\pi_{i+1} = \pi_{i'}$ or $\pi_i = \pi_{i'-1}$, and $\pi' \preceq \pi[1, i], \pi[i', len(\pi)]$ (informally speaking, π' can be obtained from π by removing a subsequence of nodes).

▶ **Fact 1.** *Let π and π' be two rides such that $\pi' \preceq \pi$. Then: $w(\pi') \leq w(\pi)$; if π' satisfies a request $(s, t) \in \mathcal{C}$, then π satisfies (s, t), too; if π is feasible (resp., optimal) and $V_{\mathcal{C}} \cap (nodes(\pi) \setminus nodes(\pi')) = \emptyset$, then π' is feasible (resp., optimal), too.*

It is easily seen that computing optimal rides is an intractable problem (**NP**-hard), for instance, by exhibiting a reduction from the well-known traveling salesman problem (see, e.g., [16]). Actually, we can strengthen this result, in a way that suggest to focus our subsequent analysis on ride-sharing scenarios over graphs whose maximum degree is 2 (at most). In fact, these graphs must be either paths or cycles.

▶ **Theorem 2.** *Computing optimal rides is* **NP***-hard over trees whose maximum degree is 3.*

3 Optimal Rides on Paths

In this section we describe an algorithm that, given as input a ride-sharing scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$ where $G = (V, E, w)$ is a *path*, returns an optimal ride for $\mathcal{R}$. In order to keep notation simple, we assume that nodes in V are (indexed as) natural numbers, so that $V = \{1, \ldots, n\}$. Hence, for each node $v \in V \setminus \{n\}$, the edge $\{v, v+1\}$ is in E; and no further edge is in E. Moreover, let us define $\mathsf{left}(\mathcal{R}) = \mathsf{min}_{v \in V_{\mathcal{C}}} v$ and $\mathsf{right}(\mathcal{R}) = \mathsf{max}_{v \in V_{\mathcal{C}}} v$, as the extreme (left and right) endpoints of any request in $\mathcal{C}$.

Based on these notions, we distinguish two mutually exclusive cases:

"outer": where either $s_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq t_0$ or $t_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq s_0$; that is, the starting and the terminating nodes s_0 and t_0 are not properly included in the range $\{\mathsf{left}(\mathcal{R}), ..., \mathsf{right}(\mathcal{R})\}$.

"inner": where $\{s_0, t_0\} \cap \{v \in V \mid \mathsf{left}(\mathcal{R}) < v < \mathsf{right}(\mathcal{R})\} \neq \emptyset$; in particular, in this case, $\mathsf{left}(\mathcal{R}) < \mathsf{right}(\mathcal{R})$ necessarily holds.

In the following two subsections we describe methods to address the two different cases, while their complexity will be later analyzed in Section 3.3. A basic ingredient for both methods is the concept of concatenation of rides. Let $\pi = \pi_1, \ldots, \pi_k$ and $\pi' = \pi'_1, \ldots, \pi'_h$ be two rides. Their *concatenation* $\pi \mapsto \pi'$ is the ride inductively defined as follows: if $\pi_k = \pi'_1$ and $h > 1$, then $\pi \mapsto \pi' = \pi_1, \ldots, \pi_k, \pi'_2, \ldots, \pi'_h$; if $\pi_k = \pi'_1$ and $h = 1$, then $\pi \mapsto \pi' = \pi$; if

Algorithm 1: RIDEONPATH_OUTER

Input: A scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$, where $G = (V, E, w)$ is a path, and with $s_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq t_0$ or $t_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq s_0$;
Output: An optimal ride for $\mathcal{R}$;

```
1 if s_0 > t_0 then
2 |   π ← RideOnPath_Outer(sym(R));
3 |   return sym(π);
4 else
5 |   C* = {(s_1, t_1), ..., (s_h, t_h)} ← Normalize(C);   /* s_1 ≤ s_2 ··· ≤ s_h */
6 |   return s_0 ↦ s_1 ↦ t_1 ↦ s_2 ↦ ... ↦ s_h ↦ t_h ↦ t_0;
```

$\pi_k \neq \pi'_1$, then $\pi \mapsto \pi'$ is defined as the concatenation[1] $\pi \mapsto \bar{\pi} \mapsto \pi'$, where $\bar{\pi} = \pi_k, \ldots, \pi'_1$ is the ride obtained as the sequence of nodes connecting π_k and π'_1 with the smallest length. Note that $\bar{\pi}$ is univocally determined on paths.

3.1 Solution to the "outer" case

Consider Algorithm 1, named RIDEONPATH_OUTER. In the first step, it distinguishes the case $s_0 > t_0$ from the case $s_0 \leq t_0$. Indeed, the former can be reduced to the latter by introducing the concept of *symmetric* scenario. For every node $v \in V$, let $\mathsf{sym}(v) = n - v + 1$. Denote by $\mathsf{sym}(\pi)$ and $\mathsf{sym}(\mathcal{C})$ the ride and the set of requests derived from the ride π and the set of requests $\mathcal{C}$, respectively, by replacing each node v with its "symmetric" counterpart $\mathsf{sym}(v)$. Finally, denote by $\mathsf{sym}(\mathcal{R})$ the scenario $\langle G, (\mathsf{sym}(s_0), \mathsf{sym}(t_0)), \mathsf{sym}(\mathcal{C}) \rangle$, referred to as the symmetric scenario of $\mathcal{R}$. Then, the following is immediately seen to hold.

▶ **Fact 3.** *Let π be a ride. Then, π is optimal for $\mathcal{R}$ if, and only if, $\mathsf{sym}(\pi)$ is optimal for $\mathsf{sym}(\mathcal{R})$.*

According to the previous observation, step 5 and step 6 are the core of the computation by addressing the case $s_0 \leq t_0$, where hence $s_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq t_0$. The idea is to reduce the set of requests $\mathcal{C}$ to an "equivalent" set of requests $\mathcal{C}^*$, which presents a simpler structure that we call *normal form*. Formally, let $\mathcal{C}^* = \{(s_1, t_1), \ldots, (s_h, t_h)\}$, and let us say that $\mathcal{C}^*$ is in normal form if $t_i < s_i$ for each $i \in \{1, \ldots, h\}$, and $s_i < t_{i+1}$ for each $i \in \{1, \ldots, h-1\}$. The reduction is performed at step 5, where NORMALIZE is invoked.

The definition of NORMALIZE is shown in Algorithm 2: Step 1 is responsible of filtering out all requests (s, t) such that $s \leq t$. Steps 2 and 3 iteratively "merge" all pairs of requests (s, t) and (s', t') such that $t < s$, $t' < s'$ and $t' \leq t \leq s' \leq s$. Finally, steps 4 and 5 remove all requests (s, t) with $t < s$ and for which there is a request (s', t') such that $t' \leq t < s \leq s'$. It can be shown that the set of requests $\mathcal{C}^*$ returned by NORMALIZE is in normal form and that the optimal ride for the ride-sharing scenario $\langle G, (s_0, t_0), \mathcal{C}^* \rangle$ is an optimal ride also for $\mathcal{R}$. In particular, it can be shown that an optimal ride for $\langle G, (s_0, t_0), \mathcal{C}^* \rangle$ can be obtained by concatenating the rides connecting s_i to t_i, incrementally from $i = 1$ to $i = h$, as it is implemented in step 6 of RIDEONPATH_OUTER. Thus, the following can be established.

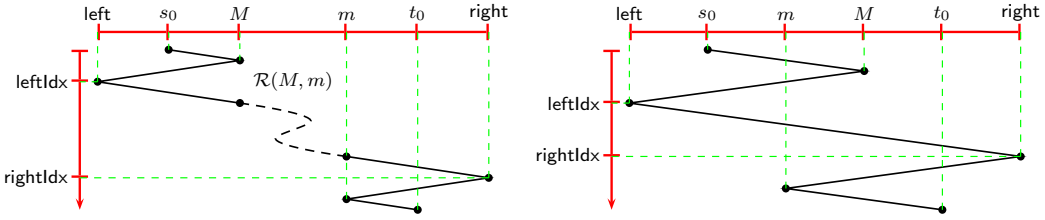
▶ **Theorem 4.** *Algorithm* RIDEONPATH_OUTER *is correct.*

[1] The specific order of application of the operator $\mapsto$ is immaterial. Hence, we often avoid the use of parenthesis.

Algorithm 2: NORMALIZE

Input: A set $\mathcal{C}$ of requests with $s_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq t_0$;
Output: A set of requests $\mathcal{C}^*$ in normal form and such that $opt(\langle G, (s_0, t_0), \mathcal{C}^* \rangle) \subseteq opt(\mathcal{R})$;

```
C* ← C \ {(s,t) | s ≤ t};
while exist (s,t),(s',t') ∈ C* such that t < s, t' < s', and t' ≤ t ≤ s' ≤ s do
    C* ← C* \ {(s,t),(s',t')} ∪ {(s,t')};
while exist (s,t),(s',t') ∈ C* such that t' ≤ t < s ≤ s' do
    C* ← C* \ {(s,t)};
return C*;
```



(a) $M < m$ (b) $m \leq M$

Figure 1 Example of (M, m)-canonical rides.

3.2 Solution to the "inner" case

Let us now move to analyze the "inner" case, where $\{s_0, t_0\} \cap \{v \in V \mid \mathsf{left}(\mathcal{R}) < v < \mathsf{right}(\mathcal{R})\} \neq \emptyset$ holds. Let us introduce some notation. For any feasible ride π, denote by $\mathsf{leftIdx}(\pi)$ (resp., $\mathsf{rightIdx}(\pi)$) the minimum time step i such that $\pi_i = \mathsf{left}(\mathcal{R})$ (resp., $\pi_i = \mathsf{right}(\mathcal{R})$). Note that $\mathsf{leftIdx}(\pi)$ and $\mathsf{rightIdx}(\pi)$ are well defined and, in particular, $\mathsf{leftIdx}(\pi) \neq \mathsf{rightIdx}(\pi)$ holds, since $\mathsf{left}(\mathcal{R}) < \mathsf{right}(\mathcal{R})$. Moveover, for every pair of nodes $x, y \in V$ with $x < y$, define $\mathcal{R}(x, y) = \langle G, (x, y), \{(s, t) \in \mathcal{C} \mid x \leq s, t \leq y\} \rangle$, that is, the scenario which inherits from $\mathcal{R}$ the graph G and every request with both starting and terminating nodes in the interval $\{x, ..., y\}$, and where the vehicle is asked to start from x and to terminate at y. Notice that, by definition, the set of all nodes occurring in any optimal ride for $\mathcal{R}(x, y)$ is a subset of $\{x, ..., y\}$.

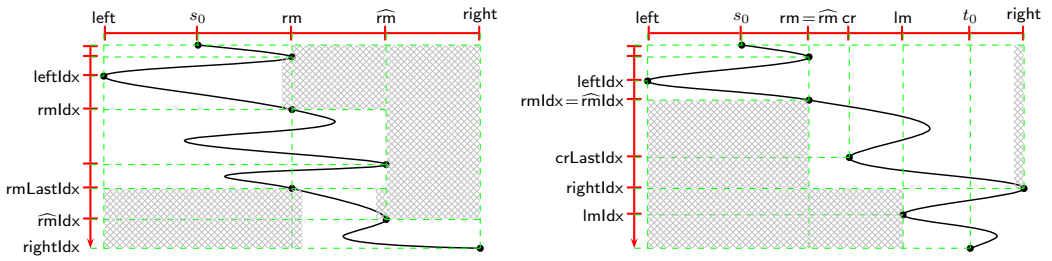
3.2.1 Canonical rides

A crucial role in our analysis is played by the concept of canonical ride, which is illustrated below.

▶ **Definition 5.** *Let $M, m \in V_{\mathcal{C}} \cup \{s_0, t_0\}$ be two nodes. A ride π^{c} in $\mathcal{R}$ is said to be* (M, m)-canonical *if $\pi^{\mathsf{c}} = \pi' \mapsto \pi'' \mapsto \pi'''$ where: $\pi' = s_0 \mapsto M \mapsto \mathsf{left}(\mathcal{R}) \mapsto M$; $\pi'' = M \mapsto \mathsf{right}(\mathcal{R})$ (resp., $\pi'' = \bar{\pi} \mapsto \mathsf{right}(\mathcal{R})$, with $\bar{\pi}$ being an optimal ride for $\mathcal{R}(M, m)$) if $m \leq M$ (resp., $m > M$); $\pi''' = \mathsf{right}(\mathcal{R}) \mapsto m \mapsto t_0$.* □

Two examples of canonical rides are in Figure 1. Note that if $m \leq M$ holds, we can refer without ambiguities to *the* (M, m)-canonical ride, as there is precisely one ride enjoying the properties in Definition 5.

▶ **Fact 6.** *If $m \leq M$, then the (M, m)-canonical ride is $s_0 \mapsto M \mapsto \mathsf{left}(\mathcal{R}) \mapsto \mathsf{right}(\mathcal{R}) \mapsto m \mapsto t_0$.*



(a) Some critical steps on a ride π^*

(b) Some critical steps on a ride π^{h+1}

Figure 2 Some critical steps of any feasible ride on a path. The gray areas denote the space that no feasible ride can cross for a given time interval.

Instead, whenever $m > M$, there can be more than one canonical ride. In this case, to compute a (M, m)-canonical ride, we need to compute an optimal ride for $\mathcal{R}(M, m)$, which is a scenario fitting the "outer" case and which can be hence addressed via the RideOnPath_Outer algorithm.

As claimed by the following theorem, the notion of canonical ride characterizes the optimal rides for $\mathcal{R}$. In particular, observe that in the following result, we focus on optimal rides π^* such that $\mathsf{leftIdx}(\pi^*) < \mathsf{rightIdx}(\pi^*)$. Indeed, the case where $\mathsf{leftIdx}(\pi^*) \geq \mathsf{rightIdx}(\pi^*)$ will be eventually addressed by working on the symmetric scenario $\mathsf{sym}(\mathcal{R})$, according to the approach discussed in Section 3.1 (see Fact 3).

▶ **Theorem 7.** *Assume that π^* is an optimal ride with $\mathsf{leftIdx}(\pi^*) < \mathsf{rightIdx}(\pi^*)$. Then, there are two nodes $M, m \in V_{\mathcal{C}} \cup \{s_0, t_0\}$, with $s_0 \leq M$ and $m \leq t_0$, such that any (M, m)-canonical ride is optimal, too.*

The proof of the result is rather involved, and we provide an overview here. We repeatedly use *exchange arguments* to gradually transform a given optimal ride without hurting its quality. Hence, let us assume that π^* is a given optimal ride such that $\mathsf{leftIdx}(\pi^*) < \mathsf{rightIdx}(\pi^*)$.

We first define a number of critical time steps and nodes of the path which are useful to analyze the properties of any optimal ride π. To help the intuition, the reader is referred to Figure 2(a). Let $\mathsf{rm}(\pi) = \mathsf{max}_{1 \leq i \leq \mathsf{leftIdx}(\pi)} \pi_i$. Note that $\mathsf{rm}(\pi) < \mathsf{right}(\mathcal{R})$ necessarily holds. Let $\mathsf{rmIdx}(\pi)$ be the minimum time step $i \geq \mathsf{leftIdx}(\mathcal{R})$ such that $\pi_i = \mathsf{rm}(\pi)$. Note that that $\mathsf{rmIdx}(\pi)$ is well defined, because $\mathsf{leftIdx}(\pi) < \mathsf{rightIdx}(\pi)$ and, hence, the ride π has to cross the node $\mathsf{rm}(\pi)$ at least once between the time step $\mathsf{leftIdx}(\pi)$ and the time step $\mathsf{rightIdx}(\pi)$. In fact, it actually holds that $\mathsf{rmIdx}(\pi) < \mathsf{rightIdx}(\pi)$, since $\mathsf{rm}(\pi) < \mathsf{right}(\mathcal{R})$. Then, define $\mathsf{rmLastIdx}(\pi)$ as the maximum time step $i \leq \mathsf{rightIdx}(\pi)$ such that $\pi_i = \mathsf{rm}(\pi)$. Note that $\mathsf{rmLastIdx}(\pi)$ coincides with $\mathsf{rmIdx}(\pi)$ if, and only if, there is no time step i such that $\mathsf{rmIdx}(\pi) < i \leq \mathsf{rightIdx}(\pi)$ with $\pi_i = \mathsf{rm}(\pi)$. Again, observe that $\mathsf{rmLastIdx}(\pi) < \mathsf{rightIdx}(\pi)$ holds. Now, define $\widehat{\mathsf{rm}}(\pi) = \mathsf{max}_{\mathsf{rmIdx}(\pi) \leq i \leq \mathsf{rmLastIdx}(\pi)} \pi_i$. Since $\mathsf{rmLastIdx}(\pi) < \mathsf{rightIdx}(\pi)$ and since $\mathsf{rightIdx}(\pi)$ is the minimum time step where the ride reaches the extreme node $\mathsf{right}(\mathcal{R})$, we have that $\widehat{\mathsf{rm}}(\pi) < \mathsf{right}(\mathcal{R})$. Moreover, $\widehat{\mathsf{rm}}(\pi) \geq \mathsf{rm}(\pi)$ clearly holds. Therefore, there is some time step between $\mathsf{rmLastIdx}(\pi)$ and $\mathsf{rightIdx}(\pi)$ where π crosses $\widehat{\mathsf{rm}}(\pi)$. So, we can define $\widehat{\mathsf{rm}}\mathsf{Idx}(\pi)$ as the minimum index $i \geq \mathsf{rmLastIdx}(\pi)$ such that $\pi_i = \widehat{\mathsf{rm}}(\pi)$, by noticing that $\widehat{\mathsf{rm}}\mathsf{Idx}(\pi) < \mathsf{rightIdx}(\pi)$ holds.

A basic transformation preserving optimality is illustrated in the following lemma.

▶ **Lemma 8.** *Assume there is an optimal ride $\pi \in opt(\mathcal{R})$ such that $\mathsf{leftIdx}(\pi) < \mathsf{rightIdx}(\pi)$. Then, the ride $s_0 \mapsto \widehat{\mathsf{rm}}(\pi) \mapsto \mathsf{left}(\mathcal{R}) \mapsto \widehat{\mathsf{rm}}(\pi) \mapsto \pi[\widehat{\mathsf{rm}}\mathsf{Idx}(\pi), len(\pi)]$ is also optimal.*

Consider now the optimal ride π^*, and the succession of optimal rides π^j, with $j \geq 0$, obtained by repeatedly applying Lemma 8. First, we set $\pi^0 = \pi^*$. Then, for each $j \geq 0$, we define π^{j+1} as the optimal ride having the form: $s_0 \mapsto \widehat{\mathsf{rm}}(\pi^j) \mapsto \mathsf{left}(\mathcal{R}) \mapsto \widehat{\mathsf{rm}}(\pi^j) \mapsto \pi^j[\widehat{\mathsf{rmIdx}}(\pi^j), len(\pi^j)]$.

In the above succession, there must exists an optimal ride π^h, with $h \geq 0$, such that $\widehat{\mathsf{rm}}(\pi^h) = \mathsf{rm}(\pi^h)$. Indeed, note that $\mathsf{rm}(\pi^{j+1}) = \widehat{\mathsf{rm}}(\pi^j)$ holds, for each $j \geq 0$, and we know that, for any optimal ride π, $\mathsf{rm}(\pi) \leq \widehat{\mathsf{rm}}(\pi) < \mathsf{right}(\mathcal{R})$. For this optimal ride π^h, we have that $\mathsf{rmLastIdx}(\pi^h) = \widehat{\mathsf{rmIdx}}(\pi^h)$, by definition of these two time steps. Therefore, $\pi^{h+1} = s_0 \mapsto \mathsf{rm}(\pi^h) \mapsto \mathsf{left}(\mathcal{R}) \mapsto \mathsf{rm}(\pi^h) \mapsto \pi^h[\mathsf{rmLastIdx}(\pi^h), len(\pi^h)]$. It is possible to prove that π^{h+1} satisfies a number of desirable properties, which can informally be summarised as follows: after time step $\widehat{\mathsf{rmIdx}}(\pi^h)$ and before reaching $\mathsf{right}(\mathcal{R})$, π^{h+1} never crosses $\widehat{\mathsf{rm}}(\pi^h)$ anymore; moreover, let $\mathsf{lm}(\pi^{h+1}) = \mathsf{min}_{\mathsf{rightIdx}(\pi^{h+1}) \leq i \leq len(\pi^{h+1})} \pi_i$, there is no request $(s,t) \in \mathcal{C}$ such that $t < \mathsf{lm}(\pi^{h+1})$, $t < \widehat{\mathsf{rm}}(\pi^h)$, and $\widehat{\mathsf{rm}}(\pi^h) < s$. For $\mathsf{lm}(\pi^{h+1}) \geq \mathsf{rm}(\pi^h)$, π^{h+1} is depicted in Figure 2(b). Note that, under such condition, π^{h+1} has an addition critical node $\mathsf{cr}(\pi^{h+1})$. Informally speaking, $\widehat{\mathsf{rm}}(\pi^h)$ is the largest node v, such that it is smaller than $\mathsf{lm}(\pi^{h+1})$ and does not admit any crossing request, i.e., a request (s,t) with $t < v \leq s$.

The properties of π^{h+1} allow us to apply a final improving transformation. In particular, we are able to show that, if $\mathsf{lm}(\pi^{h+1}) < \mathsf{rm}(\pi^h)$ then, by setting $M = \mathsf{rm}(\pi^h)$ and $m = \mathsf{lm}(\pi^{h+1})$, π^{h+1} can be reduced to the (M,m)-canonical ride depicted in Figure 2(b); otherwise, by setting $M = \mathsf{rm}(\pi^h)$ and $m = \mathsf{cr}(\pi^{h+1})$, π^{h+1} can be reduced to any of the (M,m)-canonical ride depicted in Figure 2(a).

3.2.2 An algorithm for the "inner" case

It is not difficult to see that the result in Theorem 7 immediately provides us with an algorithm to compute an optimal ride, which is based on exhaustively enumerating all possible pairs M, m of elements, by computing the associated canonical ride for each of them (either by exploiting Fact 6 if $m \leq M$, or using the RideOnPath_Outer algorithm on $\mathcal{R}(M,m)$ of $m > M$), and by eventually returning the feasible one having minimum cost. Actually, in order to deal with the case where all optimal rides π^* are such that $\mathsf{leftIdx}(\pi^*) > \mathsf{rightIdx}(\pi^*)$, we can just apply the approach over the symmetric scenario $\mathsf{sym}(\mathcal{R})$ too (see Fact 3), and return the best over the rides computed for $\mathcal{R}$ and $\mathsf{sym}(\mathcal{R})$.

Note that the approach sketched above requires the enumeration of $|V_{\mathcal{C}}|^2$ canonical rides. However, we can do better than a naïve enumeration. To this end, we explore the properties enjoyed by canonical rides that are optimal applying to the cases where $M < m$ and $M \geq m$, respectively, hold in Theorem 7.

▶ **Theorem 9.** *Assume that there are two nodes $M, m \in V_{\mathcal{C}} \cup \{s_0, t_0\}$, with $s_0 \leq M$, $m \leq t_0$ and $M < m$, such that a (M,m)-canonical ride is an optimal ride. Consider the two sets $\hat{X} = \{x \in \{s_0\} \cup V_{\mathcal{C}} \mid x \geq s_0 \ \wedge \nexists (s,t) \in \mathcal{C}$ with $t \leq x < s\}$ and $\hat{Y} = \{y \in \{t_0\} \cup V_{\mathcal{C}} \mid y \leq t_0 \ \wedge \nexists (s,t) \in \mathcal{C}$ with $t < y \leq s\}$. It holds that $\hat{X} \neq \emptyset$ and $\hat{Y} \neq \emptyset$. Moreover, let $\hat{M} = \mathsf{min}_{\hat{x} \in \hat{X}} \hat{x}$ and $\hat{m} = \mathsf{max}_{\hat{y} \in \hat{Y}} \hat{y}$, then $s_0 \leq \hat{M}$, $\hat{m} \leq t_0$, $\hat{M} < \hat{m}$ and any $(\hat{M}, \hat{m})$-canonical ride is an optimal ride, too.*

▶ **Theorem 10.** *Assume that there are two nodes $M, m \in V_{\mathcal{C}} \cup \{s_0, t_0\}$, with $s_0 \leq M$, $m \leq t_0$ and $m \leq M$, such that the (M,m)-canonical ride π^{c} is an optimal ride. Consider the set $\hat{Z}_m = \{z \in \{s_0, t_0\} \cup V_{\mathcal{C}} \mid m \leq z$ and $s_0 \leq z \ \wedge \nexists (s,t) \in \mathcal{C}$ with $t < m$ and $z < s\}$. It holds that $\hat{Z} \neq \emptyset$. Moreover, let $\hat{M}_m = \mathsf{min}_{\hat{z} \in \hat{Z}_m} \hat{z}$, then $s_0 \leq \hat{M}$, $m \leq \hat{M}_m$ and the $(\hat{M}_m, m)$-canonical ride $\hat{\pi}^{\mathsf{c}}$ is optimal, too.*

Algorithm 3: RIDEONPATH_INNER

Input: A ride-sharing scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$, where G is a path and with $\{s_0, t_0\} \cap \{v \in V \mid \mathsf{left}(\mathcal{R}) < v < \mathsf{right}(\mathcal{R})\} \neq \emptyset$;
Optionally, a Boolean value *symmetric*—set to `false`, if not provided;
Output: An optimal ride for $\mathcal{R}$;

`/* PHASE I: implementation of Theorem 9 */`
1 Compute $\hat{M}$ and $\hat{m}$, as defined in Theorem 9; // note that $\hat{M} < \hat{m}$
2 $\pi^* \leftarrow$ any $(\hat{M}, \hat{m})$-canonical ride; // use RIDEONPATH_OUTER as a subroutine for $\mathcal{R}(\hat{M}, \hat{m})$
`/* PHASE II: implementation of Theorem 10 */`
3 **for** *each node* $m \in V_{\mathcal{C}} \cup \{s_0, t_0\}$ *with* $m \leq t_0$ **do**
4 Compute $\hat{M}_m$, as defined in Theorem 10; // note that $\hat{M}_m \geq \hat{m}$
5 $\pi \leftarrow$ the $(\hat{M}_m, m)$-canonical ride; // $s_0 \mapsto \hat{M}_m \mapsto \mathsf{left}(\mathcal{R}) \mapsto \mathsf{right}(\mathcal{R}) \mapsto m \mapsto t_0$
6 **if** $w(\pi) < w(\pi^*)$ **then**
7 $\pi^* \leftarrow \pi$;
`/* PHASE III: working on the symmetric scenario */`
8 **if** *symmetric is* `false` **then**
9 $\pi^*_{\mathsf{sym}} \leftarrow$ RIDEONPATH_INNER($\mathsf{sym}(\mathcal{R})$, `true`);
10 **if** $w(\pi^*_{\mathsf{sym}}) < w(\pi^*)$ **then**
11 $\pi^* \leftarrow \mathsf{sym}(\pi^*_{\mathsf{sym}})$;
12 **return** π^*;

In the light of Theorem 7, Theorem 9 and Theorem 10, consider then Algorithm 3, named RIDEONPATH_INNER. It computes an optimal ride π^* for the "inner" case, by proceeding in three phases.

In Phase I, the algorithm computes the values $\hat{M}$ and $\hat{m}$ defined in Theorem 9 (step 1), it builds a $(\hat{M}, \hat{m})$-canonical ride, and it assigns it to π^* (step 2). Note that, according to Definition 5 and given that $\hat{M} < \hat{m}$, in order to build a $(\hat{M}, \hat{m})$-canonical ride we need to compute an optimal ride for $\mathcal{R}(\hat{M}, \hat{m})$, which is a task that we can accomplish by exploiting RIDEONPATH_OUTER as a subroutine—indeed, note that $\mathcal{R}(\hat{M}, \hat{m})$ fits the "outer" case. In Phase II, the algorithm iterates over all possible values for m in $V_{\mathcal{C}} \cup \{s_0, t_0\}$ with $m \leq t_0$. For each node m, the value $\hat{M}_m$, defined in Theorem 10, is calculated (step 4). Then, the $(\hat{M}_m, m)$-canonical ride π is built. In particular, since $\hat{M}_m \geq m$ holds, the ride π is completely determined by Fact 6. Eventually, if the cost of π is smaller than the cost of the current value of π^*, it updates π^* to π (step 7). Finally, Phase III is devoted to deal with the symmetric scenario $\mathsf{sym}(\mathcal{R})$. The idea is that the first two phases are executed again on $\mathsf{sym}(\mathcal{R})$. Let π^*_{sym} be the result of this computation (step 9). Then, we consider the symmetric ride $\mathsf{sym}(\pi^*_{\mathsf{sym}})$, which is a ride for $\mathcal{R}$, and we compare its cost with the cost of the current value of π^* (step 10). As usual, we keep the ride with the associated minimum cost, which is eventually returned as output (step 12).

Concerning the correctness, note that if $\mathcal{R}$ admits an optimal ride π with $\mathsf{leftIdx}(\pi) < \mathsf{rightIdx}(\pi)$, then by combining Theorem 7 with Theorem 9 and Theorem 10, we get that either any $(\hat{M}, \hat{m})$-canonical ride is optimal, or there is a node $m \in V_{\mathcal{C}} \cup \{s_0, t_0\}$ for which the $(\hat{M}_m, m)$-canonical ride is optimal. Instead, if every optimal ride π for $\mathcal{R}$ is such that $\mathsf{leftIdx}(\pi) > \mathsf{rightIdx}(\pi)$, then $\mathsf{sym}(\mathcal{R})$ admits an optimal ride that meets the fits the previous case. We can conclude that an optimal ride for $\mathcal{R}$ is one with the smallest cost among any $(\hat{M}, \hat{m})$-canonical ride and every $(\hat{M}_m, m)$-canonical ride, for every value of m in $V_{\mathcal{C}} \cup \{s_0, t_0\}$, both for $\mathcal{R}$ and for $\mathsf{sym}(\mathcal{R})$. Note that RIDEONPATH_INNER exhaustively searches among all the possible candidate optimal rides listed above. So, the algorithm is correct.

3.3 Implementation issues and running time

Note that checking whether an instance fits the "outer" or the "inner" case is feasible in $O(|\mathcal{C}|)$. Our goal is to show that both RideOnPath_Outer and RideOnPath_Inner can be made to run in $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$. So, we eventually establish the following result.

▶ **Theorem 11.** *Let $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$ be a ride-sharing scenario where $G = (V, E, w)$ is a path. Then, an optimal ride for $\mathcal{R}$ (together with its cost) can be computed in time $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$.*

In the implementation, we propose to sort these requests in order of starting node and, accordingly, we shall assume that $\hat{\mathcal{C}} = \{(s_1, t_1), (s_2, t_2), \ldots, (s_{|\hat{\mathcal{C}}|}, t_{|\hat{\mathcal{C}}|})\}$ holds with $s_i \leq s_j$ whenever $i < j$. Similarly, we sort the nodes in $V_{\mathcal{C}} \cup \{s_0, t_0\}$, and hence we assume that $V_{\mathcal{C}} \cup \{s_0, t_0\} = \{w_1, w_2, \ldots, w_r\}$ holds with $w_i \leq w_j$ whenever $i < j$. Moreover, for each node $w_i \in V_{\mathcal{C}} \cup \{s_0, t_0\}$, we define the set $F(w_i) = \{j \mid (s_j, t_j) \in \hat{\mathcal{C}} \wedge (w_i = s_j \text{ or } w_i = t_j)\}$, maintained as linked list. And, finally, for each element j in $F(w_i)$ we keep a label $l_{ij} \in \{\mathsf{s}, \mathsf{t}\}$ denoting whether w_i is a starting (s) or a terminating (t) node of request j. This is feasible in $O(|\mathcal{C}| \log |\mathcal{C}|)$. Given the pre-processing, it is not difficult to show that RideOnPath_Outer can be implemented in $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$, where the extra $O(|V|)$ factor comes from the need of explicitly building the ride and computing the associated cost.

Let us then analyze RideOnPath_Inner and let us focus on Phase I and Phase II, since it is immediate to check that Phase III has the same complexity.

Concerning Phase I, we first need to compute $\hat{M}$ and $\hat{m}$. To this end, we iterate through the nodes in $V_{\mathcal{C}} \cup \{s_0, t_0\}$ in order of increasing index, starting from w_1. Throughout the iteration, we maintain a set S of indexes of requests in $\hat{\mathcal{C}}$. Initially $S = \emptyset$; during the k-th iteration, we add to S every $j \in F(w_k)$ with $l_{kj} = \mathsf{t}$, and we remove from S every $j \in F(w_k)$ with $l_{kj} = \mathsf{s}$. Note that, at the end of the iteration, S contains all the elements in P_{w_k}, so that if $w_k \geq s_0$ and $S = \emptyset$, then we terminate by concluding that w_k is the smallest element in $\hat{X}$. Given the existence of $\hat{M}$, such procedure always terminates. For the complexity analysis, observe that every request in $\hat{\mathcal{C}}$ is added and removed from S exactly once. Hence, the time taken by the procedure is at most $O(|\hat{\mathcal{C}}|)$ times the maximum cost for performing each operation. If the set S is maintained as a binary min-heap, where the key of each request is its starting node, removing an element from S with label s corresponds to extract the element with smallest key, and both the insertion and the removal can be made to run in time $O(\log |\hat{\mathcal{C}}|)$. A similar approach can be used to compute $\hat{m}$. Thus, Phase I takes total time $O(|\hat{\mathcal{C}}| \log |\hat{\mathcal{C}}|)$, hence $O(|\mathcal{C}| \log |\mathcal{C}|)$, to define the pair $\hat{M}, \hat{m}$. A canonical ride with its associated cost can be then computed in $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$, since the dominant operation is the invocation of the algorithm for the outer case.

Concerning Phase II, let $m \in V_{\mathcal{C}} \cup \{s_0, t_0\}$ with $m \leq t_0$, and let $\hat{M}_m$ be the node as defined in Theorem 10. Consider the set $Q_m = \{(s', t') \in \mathcal{C} \mid t' < m < s'\}$, and let $u_m = \mathsf{max}\{m,\ s_0\}$ if $Q_m = \emptyset$, and $u_m = \mathsf{max}\{s_0,\ \mathsf{max}_{(s',t') \in Q_m} s'\}$ otherwise. Then, we can show that $\hat{M}_m = u_m$.

Therefore, for every node $w_i \in V_{\mathcal{C}} \cup \{s_0, t_0\}$, $\hat{M}_{w_i}$ is defined as the maximum between w_i and s_0, if Q_{w_i} is not empty, or the maximum between s_0 and $\mathsf{max}_{(s',t') \in Q_{w_i}} s'$, otherwise. So, the dominant operation is the computation of Q_{w_i}. To this end, for every $w_i \in V_{\mathcal{C}} \cup \{s_0, t_0\}$, we iterate through the nodes in $V_{\mathcal{C}} \cup \{s_0, t_0\}$ in order of increasing index. Note that $Q_{w_i} \subseteq \hat{\mathcal{C}}$, hence equivalently we can write $Q_{w_i} = \{(s', t') \in \hat{\mathcal{C}} \mid t' < w_i < s'\}$; this implies that, in order to compute Q_{w_i}, we need of only the requests in $\hat{\mathcal{C}}$ and we can use the usual data structures. More specifically, we iterate through the nodes in $V_{\mathcal{C}} \cup \{s_0, t_0\}$ in order of increasing index, starting from w_1. Initially, we define a set $S = \emptyset$. During the k-th iteration, we remove from S every $j \in F(w_k)$ with $l_{kj} = \mathsf{s}$, and if $k \geq 2$ we add to S every $j \in F(w_{k-1})$ with

$l_{(k-1)j} = \mathsf{t}$. Note that, at the end of the iteration, S contains all the elements in Q_{w_k}. Thus, if $S = \emptyset$, then we set M_{w_k} to $\mathsf{max}\{m, s_0\}$, otherwise we set M_{w_k} to $\mathsf{max}\{s_0, \mathsf{max}_{(s',t')\in S}\ s'\}$. In the latter case, we need to calculate $\mathsf{max}_{(s',t')\in S}\ s'$, i.e., to search in S for the request with the largest starting node. We continue in this fashion until we run out of nodes. For the complexity analysis, observe that every request in $\hat{\mathcal{C}}$ is added and removed from S exactly once. Moreover, at the end of each iteration, we need to search in S for the request with the largest starting node, in order to calculate $\mathsf{max}_{(s',t')\in S}\ s'$. Hence, the time taken by the procedure is at most $O(|\hat{\mathcal{C}}|)$ times the maximum cost for performing each operation. If the set S is maintained as a binary min-max-heap, where the key of each request is its starting node, removing an element from S with label s corresponds to extract the element with smallest key, hence both the insertion and the removal can be made to run in time $O(\log |\hat{\mathcal{C}}|)$; moreover, calculating $\mathsf{max}_{(s',t')\in S}\ s'$ corresponds to search for the element with largest key, which takes only constant time. Thus, the computation of $\hat{M}_{w_i}$, for every node $w_i \in V_{\mathcal{C}} \cup \{s_0, t_0\}$, takes a total time $O(|\hat{\mathcal{C}}| \log |\hat{\mathcal{C}}|)$, hence $O(|\mathcal{C}| \log |\mathcal{C}|)$.

Now, note that the computation of the $(\hat{M}_m, m)$-canonical ride takes constant time, since by Fact 6, we know that this ride has the form $s_0 \mapsto \hat{M}_m \mapsto \mathsf{left}(\mathcal{R}) \mapsto \mathsf{right}(\mathcal{R}) \mapsto m \mapsto t_0$. Then, the remaining operation in Phase II is the comparison between the cost of the given best ride and cost of the current ride. We have already seen that the computation of the cost of rides built in Phase I can be accommodated in the overall $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$ cost. Now, we claim that the computation of the cost of the $(\hat{M}_m, m)$-canonical ride takes constant time, provided a suitable pre-processing. Indeed, observe that the $(\hat{M}_m, m)$-canonical ride is succinctly represented by a constant number of nodes. The idea is then to associate each node $x \in V$ with the value $cw(x) = \sum_{i=2}^{x} w(\{i, i+1\})$, which is overall feasible in $O(|V|)$. Then, the cost for a rides moving from a node x to a node y, along the unique path as defined in the notion of canonical ride, is just given by the value $|cw(y) - cw(x)|$. Therefore, with a constant overhead, the cost of the $(\hat{M}_m, m)$-canonical ride can be computed. Putting it all together, Phase II can be implemented in $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$, too.

4 Optimal Rides on Cycles

In this section, we consider scenarios $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C}\rangle$ such that the underlying graph $G = (V, E, w)$, with $V = \{1, \ldots, n\}$, is a *cycle*. Formally, for each node $v \in V \setminus \{n\}$, the edge $\{v, v+1\}$ is in E; moreover, the edge $\{n, 1\}$ is in E; and no further edge is in E. Without loss of generality, we assume $s_0 = 1$.

The solution approach we shall propose is to reuse the methods we have already developed to deal with scenarios over paths. In this section, we define the key technical ingredients, and based on them an algorithm will be subsequently illustrated. Let π be a ride on $\mathcal{R}$, and let us associate each of its time steps i with a "virtual" node $\tau_\pi(i) = \pi_i + (\ell_\pi(i) - \mathsf{min}_{j\in\{1,\ldots,len(\pi)\}}\ell_\pi(j)) \cdot n$, where $\ell_\pi(1) = 0$ and where, for each $i \in \{2, \ldots, len(\pi)\}$, $\ell_\pi(i)$ is an integer defined as follows: $\ell_\pi(i) = \ell_\pi(i-1)+1$ if $\pi_{i-1} = n$ and $\pi_i = 1$; $\ell_\pi(i) = \ell_\pi(i-1)-1$ if $\pi_{i-1} = 1$ and $\pi_i = n$; and $\ell_\pi(i) = \ell_\pi(i-1)$ otherwise.

Intuitively, the function τ_π keeps track of the number of times in which the cycle is completely traversed by the ride, either clockwise or anti clockwise. Note that $\tau_\pi(i)\ \mathtt{mod}\ n = \pi_i$.

Let $\mathsf{cw}(\pi)$ (resp., $\mathsf{acw}(\pi)$) be the maximum (resp., minimum) value of $\tau_\pi(i)$ over all time steps $i \in \{1, \ldots, len(\pi)\}$. Let $\mathsf{cwIdx}(\pi)$ (resp., $\mathsf{acwIdx}(\pi)$) be the minimum time step $i \in \{1, \ldots, len(\pi)\}$ such that $\tau_\pi(i) = \mathsf{acw}(\pi)$ (resp., $\tau_\pi(i) = \mathsf{cw}(\pi)$). Note that $1 \leq \mathsf{acw}(\pi) \leq n$ always hold, by definition of τ_π. In fact, over optimal rides, useful characterizations and bounds can be derived for both $\mathsf{acw}(\pi)$ and $\mathsf{cw}(\pi)$.

Algorithm 4: RideOnCycle

Input: A ride-sharing scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$, where G is a cycle;
Output: An optimal ride for $\mathcal{R}$;

1 **for** *each tuple* $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$ *of elements as in Theorem 14* **do**
2 Let π° be an optimal ride for $\langle G^\circ, (v_{s_0}, v_{t_0}), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\} \rangle$;
3 **if** π^* *is not yet defined or* $w^\circ(\pi^\circ) < w^\circ(\pi^*)$ **then**
4 $\pi^* \leftarrow \pi^\circ$;
5 **return** $\pi^*_1 \texttt{ mod } n, \ldots, \pi^*_{len(\pi^*)} \texttt{ mod } n$;

▶ **Lemma 12.** *An optimal ride* π *exists with* $\mathsf{cw}(\pi) \leq 3n$ *and* $\{\mathsf{cw}(\pi) \texttt{ mod } n, \mathsf{acw}(\pi) \texttt{ mod } n\} \subseteq V_{\mathcal{C}} \cup \{s_0, t_0\}$.

Now, consider the path $G^\circ = (V^\circ, E^\circ, w^\circ)$, where $V^\circ = \{1, \ldots, 3n\}$ and where w° is the function such that $w^\circ(\{v, v+1\}) = w(\{v \texttt{ mod } n, (v+1) \texttt{mod } n\})$. For each pair of nodes $\alpha, \beta \in V^\circ$ with $\alpha \leq \beta$, let us define $V^\circ_{\alpha,\beta}$ as the set of nodes $v \in \{\alpha, \ldots, \beta\}$ for which no other distinct node $v' \in \{\alpha, \ldots, \beta\}$ exists such that $v \texttt{ mod } n = v' \texttt{ mod } n$. Note that if $\beta < \alpha + n$, then $V^\circ_{\alpha,\beta} = \{\alpha, \ldots, \beta\}$; if $\beta \geq \alpha + 2n - 1$, then $V^\circ_{\alpha,\beta} = \emptyset$; if $\alpha + n \leq \beta < \alpha + 2n - 1$, then $V^\circ_{\alpha,\beta} = \{\beta - n + 1, \ldots, \alpha + n - 1\}$. Moreover, define $\mathcal{C}^\circ_{\alpha,\beta} = \{(v_s, v_t) \mid (v_s \texttt{ mod } n, v_t \texttt{ mod } n) \in \mathcal{C}, v_s \in V^\circ_{\alpha,\beta}, v_t \in V^\circ_{\alpha,\beta}\}$.

▶ **Theorem 13.** *Let* π *be a feasible ride for* $\mathcal{R}$ *with* $\mathsf{cw}(\pi) \leq 3n$ *and such that* $\mathsf{acwIdx}(\pi) \leq \mathsf{cwIdx}(\pi)$ *(resp.,* $\mathsf{acwIdx}(\pi) > \mathsf{cwIdx}(\pi)$*). Let* $\alpha = \mathsf{acw}(\pi)$ *and* $\beta = \mathsf{cw}(\pi)$*, and let* $(s^\circ, t^\circ) = (\alpha, \beta)$ *(resp.,* $(s^\circ, t^\circ) = (\beta, \alpha)$*). Then, the ride* $\tau_\pi(1), ..., \tau_\pi(len(\pi))$ *is feasible for* $\langle G^\circ, (\tau_\pi(1), \tau_\pi(len(\pi))), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\} \rangle$.

Intuitively, the result tells us that feasible rides for $\mathcal{R}$ are mapped into feasible rides for a suitable defined scenario over a path. Below, we show that the converse also holds, under certain technical conditions.

▶ **Theorem 14.** *Assume that:* ***(i)*** $\alpha, \beta \in V^\circ$ *is a pair of nodes such that* $\{\alpha \texttt{ mod } n, \beta \texttt{ mod } n\} \subseteq V_{\mathcal{C}} \cup \{s_0, t_0\}$*,* $1 \leq \alpha, \beta \leq 3n$*, and such that, for each* $x \in V_{\mathcal{C}} \cup \{s_0, t_0\}$*, there is a node* $v_x \in V^\circ$ *with* $\alpha \leq v_x \leq \beta$ *and* $x = v_x \texttt{ mod } n$*.* ***(ii)*** $v_{s_0}, v_{t_0} \in V^\circ$ *is a pair of nodes such that* $\alpha \leq v_{s_0} \leq \beta$*,* $\alpha \leq v_{t_0} \leq \beta$*,* $v_{s_0} \texttt{ mod } n = s_0$*, and* $v_{t_0} \texttt{ mod } n = t_0$*.* ***(iii)*** (s°, t°) *is a request such that* $(s^\circ, t^\circ) \in \{(\alpha, \beta), (\beta, \alpha)\}$*. Let* π° *be a feasible ride for* $\langle G^\circ, (v_{s_0}, v_{t_0}), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\} \rangle$*. Then,* $\pi^\circ_1 \texttt{ mod } n, \ldots, \pi^\circ_{len(\pi^\circ)} \texttt{ mod } n$ *is a feasible ride for* $\mathcal{R}$*.*

Armed with the above technical ingredients, we can now illustrate Algorithm 4, which we refer to as RideOnCycle. This algorithm computes an optimal ride for any ride-sharing scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$, with G being a cycle. The algorithm founds on the idea of enumerating each possible tuple $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$ of elements as in Theorem 14. For each given configuration, the optimal ride π° over the scenario $\langle G^\circ, (v_{s_0}, v_{t_0}), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\} \rangle$ is computed. Eventually, π^* is defined (see step 3) as the ride with minimum cost (w.r.t. w°) over such rides π°. The ride $\pi^*_1 \texttt{ mod } n, \ldots, \pi^*_{len(\pi^*)} \texttt{ mod } n$ is then returned. Now, we claim the following.

▶ **Theorem 15.** *Let* $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$ *be a ride-sharing scenario where* $G = (V, E, w)$ *is a cycle. Then, an optimal ride for* $\mathcal{R}$ *can be computed in time* $O(|V_{\mathcal{C}}|^2 \cdot (|\mathcal{C}| \log |\mathcal{C}| + |V|))$.

In order to analyze the correctness, observe that by Theorem 14, the ride returned as output, say $\Lambda^* = \pi^*_1 \texttt{ mod } n, \ldots, \pi^*_{len(\pi^*)} \texttt{ mod } n$, is necessarily feasible for $\mathcal{R}$. Therefore,

assume for the sake of contradiction that there is an optimal ride π for $\mathcal{R}$ such that $w(\pi) < w(\Lambda^*)$. In particular, by construction of w°, we derive that $w(\pi) < w(\Lambda^*) = w^\circ(\pi^*)$. Now, by Lemma 12, we can actually assume, w.l.o.g., that $\mathsf{cw}(\pi) \leq 3n$ and $\{\mathsf{cw}(\pi) \bmod n, \mathsf{acw}(\pi) \bmod n\} \subseteq V_{\mathcal{C}} \cup \{s_0, t_0\}$ hold. So, we can apply Theorem 13 and derive the existence of a tuple $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$ of elements, with $v_{s_0} = \tau_\pi(1)$ and $v_{t_0} = \tau_\pi(len(\pi))$, satisfying properties *(i)*, *(ii)*, and *(iii)* in Theorem 14 and such that $\Upsilon = \tau_\pi(1), ..., \tau_\pi(len(\pi))$ is feasible for $\langle G^\circ, (v_{s_0}, v_{t_0}), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\} \rangle$. In particular, by construction of w°, we derive that $w^\circ(\Upsilon) = w(\pi)$. However, the algorithm has compared the weight of Υ and π^*, and hence we know that $w(\pi) = w^\circ(\Upsilon) \geq w^\circ(\pi^*)$, which is impossible.

Let us finally discuss about the implementation and running time of the algorithm. Before starting the loop, we first compute the sets $W = \{w \in V^\circ \mid 1 \leq w \leq 3n \text{ and } (w \bmod n) \in V_{\mathcal{C}} \cup \{s_0, t_0\}\}$ and $\mathcal{C}^\circ = \{(s,t) \in W \mid (s \bmod n, t \bmod n) \in \mathcal{C}\}$; this can be done in time $O(|\mathcal{C}|)$ by iterating through the requests in $\mathcal{C}$. Note that $|W| = O(|V_{\mathcal{C}}|)$ and $\mathcal{C}^\circ| = O(|\mathcal{C}|)$. Now, note that the number of iterations of RideOnCycle corresponds to the number tuples $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$ which satisfy the conditions of Theorem 14. The number of possible pairs (α, β) is $W^2 = O(|V_{\mathcal{C}}|^2)$. Checking whether condition *(i)* in Theorem 14 holds on them can be simply accomplished by checking that every element $x \in V_{\mathcal{C}} \cup \{s_0, t_0\}\}$ is such that $\alpha \bmod n \leq x \leq \beta \bmod n$. So, it can be done in constant time after that, in a pre-processing step costing $O(|V_C|)$, the minimum and maximum element in $V_{\mathcal{C}} \cup \{s_0, t_0\}\}$ have been computed. Moreover, note that since $1 \leq \alpha, \beta \leq 3n$, according to Theorem 14, there are at most 3 possible choices for s_0 (resp, t_0); in addition, there are just two alternatives for the pair s°, t°. Hence, summarizing we have that all tuples satisfying the conditions of Theorem 14 can be enumerated in $O(|V_{\mathcal{C}}|^2)$. Then, by inspecting the operations performed at each iteration, for each tuple $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$, we have to compute the set $\mathcal{C}^\circ_{\alpha,\beta}$. To this end, we search among the elements in $\mathcal{C}^\circ$ for the pairs (s,t) having both nodes in $V^\circ_{\alpha,\beta}$; this step takes $O(|\mathcal{C}|)$. Finally, on the resulting scenario defined on a path, we apply the algorithm for computing an optimal ride, which costs $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$. Hence the result in the theorem follows.

References

1. T. Asano, N. Katoh, H. Tamaki, and T. Tokuyama. Covering points in the plane by k-tours: Towards a polynomial time approximation scheme for general k. In *Proc. of STOC*, pages 275–283, 1997.
2. M.J. Atallah and S.R. Kosaraju. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, 17(5):849–869, 1988.
3. P. Chalasani and R. Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28(6):2133–2149, 1999.
4. P. Chalasani, R. Motwani, and A. Rao. Algorithms for robot grasp and delivery. In *2nd Int. Workshop on Algorithmic Foundations of Robotics*, 1996.
5. M. Charikar and B. Raghavachari. The finite capacity dial-a-ride problem. In *Proc. of FOCS*, pages 458–467, 1998.
6. M. Charikar, S. Khuller, and B. Raghavachari. Algorithms for capacitated vehicle routing. *SIAM Journal on Computing*, 31(3):665–682, 2001.
7. J.-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.
8. G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

9 Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

10 B. Eksioglu, A.V. Vural, and A. Reisman. Survey: The vehicle routing problem: A taxonomic review. *Computers and Industrial Engineering*, 57(4):1472–1483, 2009.

11 J.-F. Cordeau, G. Laporte, J.Y. Potvin, and M.W.P. Savelsbergh. Transportation on demand. In *Handbooks in operations research and management*, 2007.

12 G.N. Frederickson and D.J. Guan. Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms*, 15(1):29–60, 1993.

13 G.N. Frederickson. A note on the complexity of a simple transportation problem. *SIAM Journal on Computing*, 22(1):57–61, 1993.

14 G.N. Frederickson and D.J. Guan. Preemptive ensemble motion planning on a tree. *SIAM Journal on Computing*, 21(6):1130–1152, 1992.

15 G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

16 M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

17 M. Gendreau, G. Laporte, and D. Vigo. Heuristics for the traveling salesman problem with pickup and delivery. *Computers and Operations Research*, 26(7):699–714, 1999.

18 D.J. Guan. Routing a vehicle of capacity greater than one. *Discrete Applied Mathematics*, 81(1-3):41–57, 1998.

19 D.J. Guan and X. Zhu. Multiple capacity vehicle routing on paths. *SIAM Journal on Discrete Mathematics*, 11(4):590–602, 1998.

20 M. Haimovich and A.H.G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

21 H. Hernandez-Perez and J.J. Salazar-Gonzalez. The one-commodity pickup-and-delivery travelling salesman problem. In *Combinatorial Optimization*, pages 89–104, 2003.

22 B. Kalantari, A.V. Hill, and S.R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22(3):377–386, 1985.

23 R.M. Karp. Two combinatorial problems associated with external sorting. *Combinatorial Algorithms, Courant Computer Science Symposium 9*, pages 17–29, 1972.

24 G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

25 G. Mosheiov. Vehicle routing with pick-up and delivery: tour-partitioning heuristics. *Computers and Industrial Engineering*, 34(3):669–684, 1998.

26 J. Park and B.-I. Kim. The school bus routing problem: A review. *European Journal of Operational Research*, 202(2):311–319, 2010.

27 SophieN. Parragh, KarlF. Doerner, and RichardF. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1):21–51, 2008.

28 F. J. Srour and S. van de Velde. Are stacker crane problems easy? a statistical study. *Computers and Operations Research*, 40(3):674–690, 2013.

29 P. Toth and D. Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.

30 T.E. Tzoreff, D. Granot, F. Granot, and G. Sošić. The vehicle routing problem with pickups and deliveries on some special graphs. *Discrete Applied Mathemathics*, 116(3):193–229, 2002.