An Improved Algorithm for Open Online Dial-a-Ride*

Júlia Baligács[1 [0000−0003−2654−149X]], Yann Disser[1 [0000−0002−2085−0454]], Nils Mosis[1 [0000−0002−0692−0647]], and David Weckbecker[1 [0000−0003−3381−058X]]

TU Darmstadt, Germany
{baligacs,disser,mosis,weckbecker}@mathematik.tu-darmstadt.de

Abstract. We consider the open online dial-a-ride problem, where transportation requests appear online in a metric space and need to be served by a single server. The objective is to minimize the completion time until all requests have been served. We present a new, parameterized algorithm for this problem and prove that it attains a competitive ratio of $1 + \varphi \approx 2.618$ for some choice of its parameter, where φ is the golden ratio. This improves the best known bounds for open online dial-a-ride both for general metric spaces as well as for the real line. We also give a lower bound of 2.457 for the competitive ratio of our algorithm for any parameter choice.

Keywords: Online optimization · Dial-a-ride · Competitive analysis.

1 Introduction

In the online dial-a-ride problem, transportation requests appear over time in a metric space (M, d) and need to be transported by a single server. Each request is of the form $r = (a, b; t)$, appears at its starting position $a \in M$ at its release time $t \geq 0$, and needs to be transported to its destination $b \in M$. The server starts at a distinguished point $O \in M$, called the origin, can move at unit speed, and has a capacity $c \in \mathbb{N} \cup \{\infty\}$ that bounds the number of requests it is able to carry simultaneously. Importantly, the server only learns about request r when it appears at time t during the execution of the server's algorithm. Moreover, the total number of requests is initially unknown and the server cannot tell upon arrival of a request whether it is the last one.[1] Requests do not have to be served in the same order in which they appear.

The objective of the open dial-a-ride problem is to minimize the time until all requests have been served, by loading each request $r = (a, b; t)$ at point a no earlier than time t, transporting it to point b and unloading it there. We consider the non-preemptive variant of the problem, meaning that requests may only be unloaded at their respective destinations. Note that, in contrast to the closed

* Supported by DFG grant DI 2041/2.

[1] If the server can distinguish the last request, it can start an optimal schedule once all requests are released, achieving a completion time of at most twice the optimum.

variant of the problem, we do not require the server to return to the origin after serving the last request.

As usual, we measure the quality of a (deterministic) online algorithm in terms of competitive analysis. That is, we compare the completion time $\textsc{Alg}(\sigma)$ of the algorithm to an offline optimum completion time $\textsc{Opt}(\sigma)$ over all request sequences σ. Here, the offline optimum is given by the best possible completion time that can be achieved if all requests are known (but not released) from the start. The (strict) competitive ratio of the algorithm is given by $\rho := \inf_\sigma \textsc{Alg}(\sigma)/\textsc{Opt}(\sigma)$.[2] Note that, in particular, the running time of an algorithm does not play a role in its competitive analysis.

our results. We present a parameterized online algorithm $\textsc{Lazy}(\alpha)$ and show that it improves on the best known upper bound for open online dial-a-ride for $\alpha = \varphi$, where $\varphi = \frac{1+\sqrt{5}}{2}$ denotes the golden ratio. We also show a lower bound on potential improvements for other parameter choices. More precisely, we show the following.

Theorem 1. $\textsc{Lazy}(\varphi)$ *has competitive ratio* $1 + \varphi \approx 2.618$ *for the open online dial-a-ride problem on general metric spaces for any server capacity* $c \in \mathbb{N} \cup \{\infty\}$. *For every* $\alpha \geq 1$ *and any* $c \in \mathbb{N} \cup \{\infty\}$, $\textsc{Lazy}(\alpha)$ *has competitive ratio at least* $\max\{1 + \alpha, 2 + 2/(3\alpha)\}$, *even if the metric space is the real line.*

In particular, we obtain a lower bound on the competitive ratio of our algorithm, independent of α.

Corollary 1. *For every* $\alpha \geq 0$, $\textsc{Lazy}(\alpha)$ *has competitive ratio at least* $3/2 + \sqrt{11/12} \approx 2.457$ *for open online dial-a-ride on the line.*

Our lower bound also narrows the range of parameter choices that could allow improved competitive ratios.

Corollary 2. *For* $\alpha \notin (2\varphi/3, \varphi) \approx (1.078, 1.618)$, $\textsc{Lazy}(\alpha)$ *has competitive ratio at least* $1 + \varphi$ *for open online dial-a-ride on the line.*

Our upper bound improves the best known upper bound of 2.70 for general metric spaces [8], and even the best known upper bound of 2.67 for the real line [10]. Figure 1 gives an overview over the previous upper bounds for open online dial-a-ride. Note that, in contrast to previous results, $\textsc{Lazy}(\alpha)$ is not a so-called schedule-based algorithm as defined in [8], because it interrupts schedules.

We note that an upper bound of $\varphi + 1 \approx 2.618$ was already claimed in [26] for the Wait-or-Ignore algorithm, but the proof in [26] is inconclusive. While the general idea of our algorithm is similar to Wait-or-Ignore, our implementation is more involved and avoids issues in the analysis that are not being addressed in [26]. In particular, Wait-or-Ignore only waits at the origin, while our algorithm crucially also waits at other locations.

[2] We adopt a strict definition of the competitive ratio that requires a bounded ratio for all request sequences, i.e., we do not allow an additive constant.

related work. As listed in Figure 1, the best previously known upper bound for open online dial-a-ride of 2.70 was shown by Birx [8] and a slightly better bound of 2.67 for the line was shown by Birx et al. [10]. In this paper, we improve both bounds to $1+\varphi \approx 2.618$. A better upper bound of $1+\sqrt{2} \approx 2.41$ is known for the preemptive variant of the problem, due to Bjelde et al. [11]. The TSP problem is an important special case of dial-a-ride, where $a = b$ for every request $(a, b; t)$, i.e., requests just need to be visited. Bonifaci and Stougie gave an upper bound for open online TSP on general metric spaces of 2.41. Bjelde et al. [11] were able to show a tight bound of 2.04 for open online TSP on the line. Birx et al. [10] showed that open online dial-a-ride is strictly more difficult than open online TSP by providing a slightly larger lower bound of 2.05. Weaker lower bounds for the half-line were given by Lippmann [26].

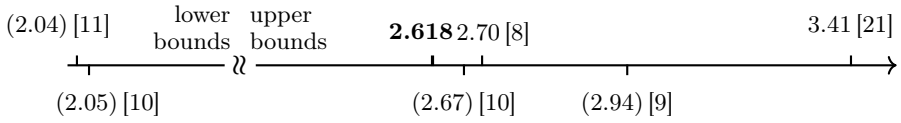

Fig. 1. Overview of the state-of-the-art for the open online dial-a-ride problem. Bounds in parentheses were shown for the real line. Note that lower bounds on the real line carry over to general metric spaces and the converse is true for upper bounds. In particular, our upper bound also holds on the line.

The competitive analysis of the closed online dial-a-ride problem on general metric spaces has proven to be structurally much simpler and conclusive results are known: The best possible competitive ratio of 2 is achieved by the conceptually clean Smartstart algorithm, as shown by Ascheuer et al. [2] and Feuerstein and Stougie [14]. Ausiello et al. [4] gave a matching lower bound already for TSP. The situation is more involved on the line. Bjelde et al. [11] gave a sophisticated algorithm for closed online TSP on the line that tightly matches the lower bound of 1.64 shown by Ausiello et al. [4]. Birx [8] separated closed online dial-a-ride on the line by giving a lower bound of 1.76. No better upper bound than 2 is known in this setting, not even for preemptive algorithms. Blom et at. [12] gave a tight bound of 1.5 for the half-line, and the best known lower bound of 1.71 for closed dial-a-ride on the half-line is due to Ascheuer et al. [2].

Clearly, most variants of the online dial-a-ride problem have resisted tight competitive analysis for many years. As a remedy, several authors have resorted to considering restricted classes of algorithms, restricted adversary models, or resource augmentation. In that vein, Blom et al. [12] considered "zealous" (or "diligent") algorithms that do not stay idle if there are unserved requests, and Birx [8] derived stronger lower bounds for "schedule-based" algorithms that subdivide the execution into schedules that may not be interrupted. Examples of restricting the adversary include "non-abusive" or "fair" models introduced by Krumke et al. [22] and Blom et al. [12], that force the optimum solution to stay

in the convex hull of all released requests. In the same spirit, Hauptmeier et al. [16] adopted a "reasonable load" model, which requires that the length of an optimum schedule for serving all requests revealed up to time t is bounded by a function of t. In terms of resource augmentation, Allulli et al. [1] and Ausiello et al. [5] considered a model with "lookahead", where the algorithm learns about requests before they are released. In contrast, Lippmann et al. [25] considered a restricted information model where the server learns the destination of a request only upon loading it. Bonifaci and Stougie [13] and Jaillet and Wagner [19] considered resource augmentation regarding the number of servers, their speeds, and their capacities.

While we concentrate on minimizing completion time, other objectives have been studied: Krumke [21] presented first results for randomized algorithms minimizing expected completion time, Krumke et al. [24] and Bienkowski et al. [6,7] minimized the sum of completion times, Krumke et al. [23,22] and Hauptmeier et al. [16] minimized the flow time, and Yi and Tian [27] maximized the number of served requests (with deadlines). Regarding other metric spaces, Jawgal et al. [20] considered online TSP on a circle. Various generalizations of online dial-a-ride have been investigated: Ausiello et al. [3] introduced the online quota TSP, where only a minimum weighted fraction of requests need to be served, and, similarly, Jaillet and Lu [17,18] adopted a model where requests can be rejected for a penalty in the objective. Jaillet and Wagner [19] and Hauptmeier et al. [15] allowed precedence constraints between requests.

2 Notation and definition of the algorithm

Let $\sigma = (r_1, \ldots, r_n)$ be a sequence of requests $r_i = (a_i, b_i; t_i)$ with release times $0 < t_1 < \cdots < t_n$. Note that we do not allow multiple requests to appear at the same time or a request to appear at time 0 but this is not a restriction as the release times can differ by arbitrarily small values. We let $\textsc{Opt}(t)$ denote the completion time of the offline optimum over all requests released not later than t. A *schedule* is a sequence of actions of the server, specifying when requests are collected and unloaded, how the server moves, and, in particular, when the server stays stationary. Let $\textsc{Opt}[t]$ denote an optimal schedule with completion time $\textsc{Opt}(t)$. We say that a server *visits* point $p \in M$ at time $t \geq 0$ if the server is in position p at time t.

The rough idea of our algorithm is to wait until we gather several requests and then start a schedule serving them. If a new request arrives during the execution of a schedule, it would be desirable to include it in the server's plan. Therefore, we check whether we can "reset" the server's state in a reasonable time, i.e., deliver all currently loaded requests and return to the origin, so that we can compute a new schedule. If this is not possible, we keep following the current schedule and consider the new requests later.

We introduce some notation to capture this more formally. Let R be a set of requests and $x \in M$. Then, the schedule $S(R, x)$ is the shortest schedule starting from point x and serving all requests in R. Note that this schedule can ignore

the release times of the requests as we will only compute it after all requests in R are released. As it is not beneficial to wait at some point during the execution of a schedule, the walked distance in $S(R, x)$ is the same as the time needed to complete it. We denote its length by $|S(R, x)|$.

Now, we can describe our algorithm. The factor $\alpha \geq 1$ will be a measure of how long we wait before starting a schedule. A precise description of the algorithm is given below (cf. Algorithm 1). In short, whenever a new request $r = (a, b; t)$ arrives, we determine whether it is possible to serve all loaded requests and return to the origin in time $\alpha \cdot \textsc{Opt}(t)$. If this is possible, we do so. In this case, we say that the schedule was *interrupted*. Otherwise, we ignore the request and consider it in the next schedule. Before starting a new schedule, we wait at least until time $\alpha \cdot \textsc{Opt}(t)$.

In the following, the algorithm $\textsc{Lazy}(\alpha)$ with waiting parameter $\alpha \geq 1$ is described. The first part of the algorithm is invoked whenever a new request $r = (a, b; t)$ is released, and the second part of the algorithm is invoked whenever the algorithm becomes idle, i.e., when the server has finished waiting or finished a schedule. We denote by t the current time, by R_t the set of unserved requests at time t and by p_t the position of the server at time t. There are three commands that can be executed, namely DELIVER_AND_RETURN, WAIT_UNTIL(t'), and FOLLOW_SCHEDULE(S). Whenever one of these commands is invoked, the server aborts what it is currently doing and executes the new command. The command DELIVER_AND_RETURN instructs the server to deliver all loaded requests and return to the origin in an optimal way. The command WAIT_UNTIL(t') orders the server to remain at its position until time t' and the command FOLLOW_SCHEDULE(S) tells the server to execute schedule S. Once the server completes the execution of a command, it becomes *idle*.

Algorithm 1: $\textsc{Lazy}(\alpha)$

initialize: $i \leftarrow 0$

upon receiving request $r = (a, b; t)$*:*
if *server can serve loaded requests and return to* O *until time* $\alpha \cdot \textsc{Opt}(t)$ **then**
  execute DELIVER_AND_RETURN `// interrupt` $S^{(i)}$

upon becoming idle:
if $t < \alpha \cdot \textsc{Opt}(t)$ **then**
  execute WAIT_UNTIL($\alpha \cdot \textsc{Opt}(t)$)
else if $R_t \neq \emptyset$ **then**
  $i \leftarrow i + 1$, $R^{(i)} \leftarrow R_t$, $t^{(i)} \leftarrow t$, $p^{(i)} \leftarrow p_t$
  $S^{(i)} \leftarrow S(R^{(i)}, p^{(i)})$
  execute FOLLOW_SCHEDULE($S^{(i)}$)

3 Analysis of LAZY

In this section, we analyze LAZY(α) and show that LAZY(α) is $1+\alpha$ competitive for $\alpha \geq \varphi = \frac{1+\sqrt{5}}{2}$. This implies in particular that LAZY(φ) is $(1+\varphi)$-competitive, i.e., that the first part of Theorem 1 holds.

Theorem 2. *For $\alpha \geq \varphi \approx 1.618$, LAZY($\alpha$) is $(1+\alpha)$-competitive for the open dial-a-ride problem on general metric spaces for any server capacity $c \in \mathbb{N} \cup \{\infty\}$.*

Proof. For a given request sequence $(r_1, \ldots, r_n)$, we denote the number of schedules started by LAZY(α) by $k \leq n$. Let $S^{(i)}$, $t^{(i)}$, $p^{(i)}$, and $R^{(i)}$ be as defined in the algorithm, i.e., $S^{(i)}$ is the i-th schedule started by LAZY(α), $t^{(i)}$ is its starting time, $p^{(i)}$ its starting position, and $R^{(i)}$ is the set of requests served by $S^{(i)}$. Observe that some schedules might be interrupted so that $R^{(1)}, \ldots, R^{(k)}$ are not necessarily disjoint. Also observe that we have $p_1 = O$ and, for $i > 1$, $p^{(i)}$ is either the ending position of $S^{(i-1)}$ or O if $S^{(i-1)}$ was interrupted.

We show by induction on i that, for all $i \in \{1, \ldots, k\}$,

a) $|S^{(i)}| \leq \text{OPT}(t^{(i)})$, and
b) $t^{(i)} + |S^{(i)}| \leq (1+\alpha) \cdot \text{OPT}(t^{(i)})$.

Note that this completes the proof since the last schedule is completed at time $t^{(k)} + |S^{(k)}|$ and since $\text{OPT}(t^{(k)})$ is the completion time of the offline optimum over all requests.

Before starting the induction, let us make some observations. Since the server does not start a schedule at time t if $t < \alpha \cdot \text{OPT}(t)$, we have

$$t^{(i)} \geq \alpha \cdot \text{OPT}(t^{(i)}) \tag{1}$$

for all $i \in \{1, \ldots, k\}$. Further, for every request $r = (a, b; t) \in R^{(i+1)} \setminus R^{(i)}$, we have $t > t^{(i)}$ because $R^{(i)}$ contains all unserved requests released until time $t^{(i)}$. Moreover, $R^{(i+1)} \setminus R^{(i)} \neq \emptyset$ because otherwise $S^{(i)}$ is not interrupted and we have $R^{(i+1)} = \emptyset$, contradicting that the algorithm starts $S^{(i+1)}$. Therefore, for all $i \in \{1, \ldots, k-1\}$,

$$\text{OPT}(t^{(i+1)}) > t^{(i)} \overset{(1)}{\geq} \alpha \cdot \text{OPT}(t^{(i)}). \tag{2}$$

Now, let us start the induction.
Base case: a) Since $\text{OPT}[t^{(1)}]$ is a schedule serving all requests in $R^{(1)}$ starting from O, possibly with additional waiting times, we have

$$|S^{(1)}| = |S(R^{(1)}, p^{(1)})| = |S(R^{(1)}, O)| \leq \text{OPT}(t^{(1)}).$$

b) Consider the time $t^{(1)}$ at which schedule $S^{(1)} = S(R^{(1)}, O)$ is started, and let $t' \leq t^{(1)}$ denote the largest release time of a request in $R^{(1)}$. In particular, no requests are released in the time period $(t', t^{(1)}]$ and thus $\text{OPT}(t') = \text{OPT}(t^{(1)})$. When the request at time t' is released, the server is in O so that the command DELIVER_AND_RETURN is completed immediately. Therefore, the server

becomes idle at time t' and the waiting time is set to $\alpha \cdot \textsc{Opt}(t') = \alpha \cdot \textsc{Opt}(t^{(1)})$. Observe that $\alpha \cdot \textsc{Opt}(t') > \textsc{Opt}(t') \geq t'$. The server becomes idle again and starts schedule $S^{(1)}$ precisely at time $t^{(1)} = \alpha \cdot \textsc{Opt}(t^{(1)})$. We obtain

$$t^{(1)} + |S^{(1)}| \overset{\text{a)}}{\leq} (1+\alpha) \cdot \textsc{Opt}(t^{(1)}).$$

Induction step: Assume that a) and b) hold for some $i \in \{1, \ldots, k-1\}$. We show that this implies that a) and b) also hold for $i+1$.

First, consider the case that the schedule $S^{(i)}$ is interrupted. Then, we have $p^{(i+1)} = O$ and $t^{(i+1)} = \alpha \cdot \textsc{Opt}(t^{(i+1)})$. It immediately follows that $S^{(i+1)} = |S(R^{(i+1)}, p^{(i+1)})| \leq \textsc{Opt}(t^{(i+1)})$ because $\textsc{Opt}[t^{(i+1)}]$ serves all requests in $R^{(i+1)}$ (among others) and starts in O. With this, we obtain

$$t^{(i+1)} + |S^{(i+1)}| \leq (1+\alpha) \cdot \textsc{Opt}(t^{(i+1)}).$$

Therefore, a) and b) hold for $i+1$ if $S^{(i)}$ is interrupted. For the rest of the proof, assume that $S^{(i)}$ is not interrupted.

Assume that $\textsc{Opt}[t^{(i+1)}]$ visits $p^{(i+1)}$ before collecting any request in $R^{(i+1)}$. Then, by definition of $S^{(i+1)} = S(R^{(i+1)}, p^{(i+1)})$, we immediately see that a) holds for $i+1$ because $\textsc{Opt}[t^{(i+1)}]$ needs to serve all requests in $R^{(i+1)}$ after visiting point $p^{(i+1)}$. Thus, for the proof of a), it suffices to consider the case that $\textsc{Opt}[t^{(i+1)}]$ collects some request in $R^{(i+1)}$ before visiting $p^{(i+1)}$. We denote the first request in $R^{(i+1)}$ collected by $\textsc{Opt}[t^{(i+1)}]$ by $r = (a, b; t)$.

Since $S^{(i)}$ is not interrupted, we have $R^{(i+1)} \cap R^{(i)} = \emptyset$ and thus $t > t^{(i)}$. Together with (1), this implies that $\textsc{Opt}[t^{(i+1)}]$ collects r at a not earlier than time $t > t^{(i)} \geq \alpha \cdot \textsc{Opt}(t^{(i)})$. By definition of r and $S(R^{(i+1)}, a)$, this implies

$$\textsc{Opt}(t^{(i+1)}) \geq t + |S(R^{(i+1)}, a)| > \alpha \cdot \textsc{Opt}(t^{(i)}) + |S(R^{(i+1)}, a)|. \tag{3}$$

Further, since we assumed that $\textsc{Opt}[t^{(i+1)}]$ visits $p^{(i+1)}$ after visiting a later than $\alpha \cdot \textsc{Opt}(t^{(i)})$ and since the server needs at least time $d(a, p^{(i+1)})$ to get from a to $p^{(i+1)}$, we have

$$\textsc{Opt}(t^{(i+1)}) \geq \alpha \cdot \textsc{Opt}(t^{(i)}) + d(a, p^{(i+1)}). \tag{4}$$

Let $t_\ell \leq t^{(i+1)}$ denote the largest release time of a request in $R^{(i+1)}$. No requests appears in the, possibly empty, time interval $(t_\ell, t^{(i+1)}]$. Thus, we have $\textsc{Opt}(t_\ell) = \textsc{Opt}(t^{(i+1)})$. Recall that the schedule $S^{(i)}$ is not interrupted. In particular, it is not interrupted at time t_ℓ, i.e., at time t_ℓ, the server cannot serve all loaded requests and return to the origin until time $\alpha \cdot \textsc{Opt}(t_\ell)$. At time t_ℓ, the server can trivially serve all loaded requests in time $t^{(i)} + |S^{(i)}|$ by following the current schedule, which ends in $p^{(i+1)}$. This yields

$$t^{(i)} + |S^{(i)}| + d(p^{(i+1)}, O) > \alpha \cdot \textsc{Opt}(t_\ell) = \alpha \cdot \textsc{Opt}(t^{(i+1)}). \tag{5}$$

Recall that $p^{(i+1)}$ is the ending position of $S^{(i)}$ and, therefore, it is the destination of a request in $R^{(i)}$. Since $\textsc{Opt}[t^{(i)}]$ needs to visit $p^{(i+1)}$ and starts in O, we

have $d(p^{(i+1)}, O) \leq \textsc{Opt}(t^{(i)})$. Further, by the induction hypothesis, we have $t^{(i)} + |S^{(i)}| \leq (1+\alpha) \cdot \textsc{Opt}(t^{(i)})$. This yields

$$\begin{aligned}(2+\alpha) \cdot \textsc{Opt}(t^{(i)}) &\geq t^{(i)} + |S^{(i)}| + d(p^{(i+1)}, O) \\ &\overset{(5)}{>} \alpha \cdot \textsc{Opt}(t^{(i+1)}) \\ &\overset{(4)}{\geq} \alpha^2 \cdot \textsc{Opt}(t^{(i)}) + \alpha \cdot d(a, p^{(i+1)}),\end{aligned}$$

so that

$$d(a, p^{(i+1)}) < \frac{1}{\alpha} \cdot (2 + \alpha - \alpha^2) \cdot \textsc{Opt}(t^{(i)}). \tag{6}$$

The schedule $S^{(i+1)}$ starts in $p^{(i+1)}$ and needs to serve all requests in $R^{(i+1)}$. By applying the triangle inequality, we can conclude that

$$\begin{aligned}|S^{(i+1)}| &\leq d(p^{(i+1)}, a) + |S(R^{(i+1)}, a)| \\ &\overset{(3)}{<} d(p^{(i+1)}, a) + \textsc{Opt}(t^{(i+1)}) - \alpha \cdot \textsc{Opt}(t^{(i)}) \\ &\overset{(6)}{<} \left(\frac{2}{\alpha} + 1 - 2\alpha\right) \cdot \textsc{Opt}(t^{(i)}) + \textsc{Opt}(t^{(i+1)}) \\ &\leq \textsc{Opt}(t^{(i+1)}),\end{aligned}$$

where the last inequality holds because we have $\left(\frac{2}{\alpha} + 1 - 2\alpha\right) \leq 0$ if $\alpha \geq \frac{1+\sqrt{17}}{4} \approx 1.2808$.

It remains to show that b) also holds for $i+1$. If the schedule $S^{(i)}$ is completed before time $\alpha \cdot \textsc{Opt}(t^{(i+1)})$, the schedule $S^{(i+1)}$ is started precisely at time $t^{(i+1)} = \alpha \cdot \textsc{Opt}(t^{(i+1)})$. Together with part a), this yields the assertion. Therefore, assume that $S^{(i)}$ is not completed before time $\alpha \cdot \textsc{Opt}(t^{(i+1)})$. Then, the schedule $S^{(i+1)}$ is started as soon as $S^{(i)}$ is completed. Together with the induction hypothesis, this implies $t^{(i+1)} = t^{(i)} + |S^{(i)}| \leq (1+\alpha) \cdot \textsc{Opt}(t^{(i)})$. Hence, the schedule $S^{(i+1)}$ can be completed in time

$$\begin{aligned}t^{(i+1)} + |S^{(i+1)}| &\leq (1+\alpha) \cdot \textsc{Opt}(t^{(i)}) + |S^{(i+1)}| \\ &\overset{(2),\,a)}{\leq} (1+\alpha) \cdot \frac{1}{\alpha} \cdot \textsc{Opt}(t^{(i+1)}) + \textsc{Opt}(t^{(i+1)}) \\ &= \left(\frac{1}{\alpha} + 2\right) \cdot \textsc{Opt}(t^{(i+1)}) \\ &\leq (1+\alpha) \cdot \textsc{Opt}(t^{(i+1)}),\end{aligned}$$

where the last inequality holds because we have $\frac{1}{\alpha} + 2 \leq 1 + \alpha$ if $\alpha \geq \frac{1+\sqrt{5}}{2} = \varphi$.

4 Lower bound for Lazy

In this section, we provide lower bounds on the competitive ratio of $\textsc{Lazy}(\alpha)$. We give a lower bound construction for $\alpha \geq 1$ and a separate construction for

$\alpha < 1$. Together they show that LAZY(α) cannot be better than $(3/2+\sqrt{11/12})$-competitive for all $\alpha \geq 0$, i.e., that Corollary 1 holds. Furthermore, they narrow the range of parameter choices that would lead to an improvement over the competitive ratio of $\varphi + 1$.

In the following constructions, we let the metric space (M, d) be the real line, i.e., $M = \mathbb{R}$, $O = 0$, and $d(a, b) = |a - b|$. Note that lower bounds on the line trivially carry over to general metric spaces. Moreover, our constructions work for any given server capacity $c \in \mathbb{N} \cup \{\infty\}$ because a larger server capacity does neither change the behavior of the optimum solution nor the behavior of LAZY.

First, observe that, for any $\alpha \geq 0$, LAZY(α) has a competitive ratio of at least $1 + \alpha$. This can be easily seen by observing the request sequence consisting of the single request $r_1 = (1, 1; \frac{1}{2})$. In this case, the offline optimum has completed the sequence by time 1, whereas LAZY(α) waits in O until time $\max(\alpha, \frac{1}{2})$ and then moves to 1 and serves r_1 not earlier than $1 + \alpha$.

Lemma 1. *For any $\alpha \geq 0$,* LAZY(α) *has a competitive ratio of at least $1+\alpha$ for the open online dial-a-ride problem on the line for any capacity $c \in \mathbb{N} \cup \{\infty\}$.*

Now, we give a construction for the case $\alpha \geq 1$.

Proposition 1. *For $\alpha \geq 1$,* LAZY(α) *has a competitive ratio of at least $2 + \frac{2}{3\alpha}$ for the open online dial-a-ride problem on the line.*

Proof. First, observe that, for $\alpha \geq 1/2 + \sqrt{11/12}$, we have $2 + \frac{2}{3\alpha} \leq 1 + \alpha$ so that the assertion follows from Lemma 1. Therefore, let $\alpha \in [1, 1/2 + \sqrt{11/12})$ and let $\varepsilon > 0$ be small enough such that $3\alpha + 2 > 3\alpha^2 + \alpha\varepsilon$. Note that this is possible because $3\alpha + 2 > 3\alpha^2$ for $\alpha \in [1, (1/2) + \sqrt{11/12})$.

We construct an instance of the open online dial-a-ride problem, where the competitive ratio of LAZY(α) converges to $2 + \frac{2}{3\alpha}$ for $\varepsilon \to 0$ (cf. Figure 2). We define the instance by giving the requests

$$r_1 = (0, 1; \varepsilon),\ r_2 = (0, -1; 2\varepsilon), \text{ and } r_3 = (2 - 3\alpha - \varepsilon, 2 - 3\alpha - \varepsilon; 3\alpha + \varepsilon).$$

One solution is to first serve r_1 and then r_2. This is possible in 3 time units and, after this, the server is in position -1. Then, the server can reach point $2-3\alpha-\varepsilon$ by time $3 + (3\alpha + \varepsilon - 2 - 1) = 3\alpha + \varepsilon$. At this point in time, r_3 is released and can immediately be served. Thus, we have

$$\text{OPT} := \text{OPT}(3\alpha + \varepsilon) = 3\alpha + \varepsilon.$$

We now analyze what LAZY(α) does on this request sequence. We have $\text{OPT}(2\varepsilon) = 3$. Thus, the server waits in O until time 3α. Since no new request arrives until this time, the server starts an optimal schedule serving r_1 and r_2. Without loss of generality, we can assume that LAZY(α) starts by serving r_2, because the starting positions and destinations of r_1 and r_2 are symmetrical. At time $3\alpha + \varepsilon$, request r_3 is released, and the server has currently loaded r_2. Delivering r_2 and returning to the origin takes the server until time $3\alpha + 2$. By definition of α and ε, we have

$$3\alpha + 2 > 3\alpha^2 + \alpha\varepsilon = \alpha\text{OPT}. \tag{7}$$

This implies that the server is not interrupted in its current schedule. It continues serving r_2 and then serves r_1 at time $3\alpha + 3$. Together with (7), it follows that, after serving r_1, the server immediately starts serving the remaining request r_3. Moving from 1 to $2 - 3\alpha - \varepsilon$ takes $3\alpha - 1 + \varepsilon$ time units, i.e., the server serves r_3 at time $(3\alpha + 3) + (2 - 3\alpha - \varepsilon) = 6\alpha + 2 + \varepsilon$. Thus, the competitive ratio is at least

$$\frac{6\alpha + 2 + \varepsilon}{\textsc{Opt}} = \frac{6\alpha + 2 + \varepsilon}{3\alpha + \varepsilon} = 2 + \frac{2 - \varepsilon}{3\alpha + \varepsilon}.$$

The statement follows by taking the limit $\varepsilon \to 0$.

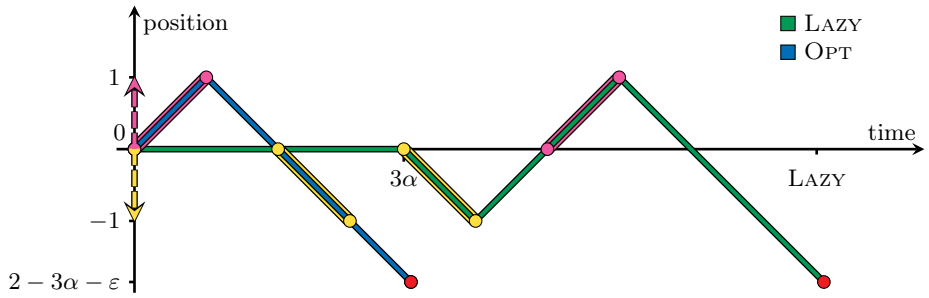

Fig. 2. Instance of the open online dial-a-ride problem on the line where $\textsc{Lazy}(\alpha)$ has a competitive ratio of at least $2 + \frac{2}{3\alpha}$ for all $\alpha \geq 1$.

Next, we give a lower bound construction for $\alpha < 1$.

Proposition 2. *For $\alpha \in [0, 1)$, the algorithm $\textsc{Lazy}(\alpha)$ has a competitive ratio of at least $1 + \frac{3}{\alpha+1}$ for the open dial-a-ride problem on the line.*

Proof. Let $\alpha \in [0, 1)$ and $\varepsilon \in (0, \min\{\frac{\alpha}{2}, \frac{1}{\alpha} - \alpha, 1 - \alpha\})$. We construct an instance of the open dial-a-ride problem, where the competitive ratio of $\textsc{Lazy}(\alpha)$ converges to $1 + \frac{3}{\alpha+1}$ for $\varepsilon \to 0$ (cf. Figure 3). We define the instance by giving the requests

$$r_1 = \left(\frac{\varepsilon}{2}, \frac{1}{2}; \frac{\varepsilon}{2}\right),\ r_2 = (1, 1; \varepsilon),\ r_3 = (0, 0; \alpha + \varepsilon),$$
$$r_4 = \left(\frac{1}{2} + \varepsilon, 1; \alpha + 2\varepsilon\right), \text{ and } r_5 = (1, 1; \alpha + 1 + \varepsilon).$$

One solution is to first wait in O until time $\alpha + \varepsilon$ and serve r_3. Then, the server can move to $\varepsilon/2$, pick up r_1 and deliver it. Then, we can move to $\frac{1}{2} + \varepsilon$, pick up r_4 and deliver it. This can be done by time $\alpha + 1 + \varepsilon$. Now, the server is in position 1 and can thus immediately serve r_5. It finishes serving all request in time $\alpha + 1 + \varepsilon$. Since the last request is released at time $\alpha + 1 + \varepsilon$, we have

$$\textsc{Opt} := \textsc{Opt}(\alpha + 1 + \varepsilon) = \alpha + 1 + \varepsilon. \tag{8}$$

We now analyze what Lazy(α) does on this request sequence. We have Opt($\varepsilon/2$) $= 1/2$ so that $\alpha \cdot \text{Opt}(\varepsilon/2) = \alpha/2 > \varepsilon$ and the server does not start moving before r_2 is released. Then, we have Opt(ε) $= 1$. Hence, the server waits in O until time α. Since no new requests arrive until this time, the server starts an optimal schedule serving r_1 and r_2, i.e., it moves to $\varepsilon/2$ and picks up r_1. At times $\alpha + \varepsilon$ and $\alpha + 2\varepsilon$, r_3 and r_4 are released. We have Opt($\alpha + \varepsilon$) $=$ Opt($\alpha + 2\varepsilon$) $= \alpha + 1 + \varepsilon$. Serving the loaded request r_1 and returning to 0 would take the server until time

$$\alpha + 1 \overset{\varepsilon < \frac{1}{\alpha} - \alpha}{>} \alpha + (\alpha^2 + \alpha\varepsilon) = \alpha(\alpha + 1 + \varepsilon) = \alpha \cdot \text{Opt}(\alpha + \varepsilon). \tag{9}$$

Thus, the server keeps following its tour and serves r_1 and then r_2 at time $\alpha+1$. By (9) and since Opt($\alpha+1$) $=$ Opt($\alpha+\varepsilon$), the server immediately starts serving r_3 and r_4. The shortest tour is serving r_4 first, i.e., the server starts moving towards $\frac{1}{2} + \varepsilon$. At time $\alpha + 1 + \varepsilon$, request r_5 is released. Since

$$\alpha + 1 + \varepsilon \overset{(8)}{=} \text{Opt}(\alpha + 1 + \varepsilon) > \alpha \cdot \text{Opt}(\alpha + 1 + \varepsilon),$$

the server keeps following its tour, which is finished at time $(\alpha + 1) + (1 - 2\varepsilon) + 1 = 3 + \alpha - 2\varepsilon$ in position O. Then, the server starts its last tour in order to serve r_5. It moves to 1 and finishes serving the last request at time $4 + \alpha - 2\varepsilon$. Thus, the competitive ratio is

$$\frac{4 + \alpha - 2\varepsilon}{\text{Opt}} = \frac{4 + \alpha - 2\varepsilon}{\alpha + 1 + \varepsilon} = 1 + \frac{3 - 3\varepsilon}{\alpha + 1 + \varepsilon}.$$

The statement follows by taking the limit $\varepsilon \to 0$.

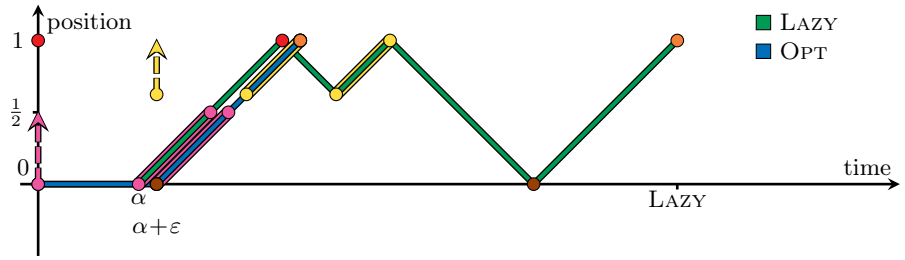

Fig. 3. Instance of the open online dial-a-ride problem on the line where Lazy(α) has a competitive ratio of at least $1 + \frac{3}{\alpha+1}$ for all $\alpha \in [0, 1)$.

We now give another lower bound construction which is stonger than the previous one for large $\alpha < 1$.

Proposition 3. *For $\alpha \in [0, 1)$, the algorithm* Lazy(α) *has a competitive ratio of at least $2 + \alpha + \frac{1-\alpha}{2+3\alpha}$ for the open online dial-a-ride problem on the line.*

Proof. Let $\varepsilon > 0$ be small enough. We construct an instance of the open online dial-a-ride problem on the line, where the competitive ratio of LAZY(α) converges to $2 + \alpha + \frac{1-\alpha}{2+3\alpha}$ for $\varepsilon \to 0$ (cf. Figure 4). We distinguish between three cases.

Case 1 ($\alpha \in [0, 2/3)$): We define the instance by giving the requests

$$\begin{aligned} r_1 &= (0, 1; \varepsilon), \\ r_2 &= (-\alpha, -\alpha; \alpha + \varepsilon), \\ r_3 &= (2 + \alpha - \varepsilon, 2 + \alpha - \varepsilon; \alpha + 2\varepsilon), \\ r_4 &= (2 + \alpha - \varepsilon, 2 + \alpha - \varepsilon; 2 + 3\alpha). \end{aligned}$$

One solution is to move to $-\alpha$ and wait there until time $\alpha + \varepsilon$. Then, r_2 can be served, and the server can move to 0 where it arrives at time $2\alpha + \varepsilon$. It picks up r_1 and delivers it at time $2\alpha + 1 + \varepsilon$ at 1. It keeps moving to position $2 + \alpha - \varepsilon$ and serves r_3 and r_4 there at time $2 + 3\alpha$. Since the last request is released at time $2 + 3\alpha$, we have

$$\text{OPT} := \text{OPT}(2 + 3\alpha) = 2 + 3\alpha.$$

We now analyze what LAZY(α) does. We have $\text{OPT}(\varepsilon) = 1 + \varepsilon$. Thus, the server starts waiting in 0 until time $\alpha(1 + \varepsilon)$. At time $\alpha(1 + \varepsilon)$, the server starts an optimal schedule over all unserved requests, i.e., over $\{r_1\}$. It picks up r_1 and starts moving towards 1. At time $\alpha + \varepsilon > \alpha(1 + \varepsilon)$, request r_2 arrives. We have $\text{OPT}(\alpha + \varepsilon) = 2\alpha + 1 + \varepsilon$. Serving the loaded request r_1 and returning to the origin would take the server until time

$$\alpha(1 + \varepsilon) + 2 > \alpha(2\alpha + 1 + \varepsilon) = \alpha\text{OPT}(\alpha + \varepsilon),$$

i.e., the server continues its current schedule. At time $\alpha + 2\varepsilon$, r_3 is released. We have $\text{OPT}(\alpha + 2\varepsilon) = 2 + 3\alpha$. Serving the loaded request and returning to the origin would still take until time

$$\alpha(1 + \varepsilon) + 2 > 2\alpha + 3\alpha^2 = \alpha\text{OPT}(\alpha + 2\varepsilon),$$

where the inequality follows from the fact that $\alpha < \frac{2}{3}$. Thus, the server continues the current schedule, which is finished at time $\alpha(1+\varepsilon)+1$ in position 1. The server waits until time $\max\{1 + \alpha, \alpha\text{OPT}(\alpha + 2\varepsilon)\}$ and then starts the next schedule. Since $\alpha\text{OPT}(\alpha+2\varepsilon) = 2\alpha+3\alpha^2 < 2+3\alpha$, the next schedule is thus started before r_4 is released. It is faster to serve r_3 before r_2 in this schedule because the server starts from point 1. At time $2 + 3\alpha$, request r_4 is released. Since this does not change the completion time of the optimum and because the server started the current schedule not earlier than time $\alpha\text{OPT}(\alpha + 2\varepsilon)$, it continues the current schedule. The second schedule takes $(1 + \alpha - \varepsilon) + (2 + 2\alpha - \varepsilon) = 3 + 3\alpha - 2\varepsilon$ time units and ends in $-\alpha$. The last schedule, in which r_4 is served, is started immediately and takes $2 + 2\alpha - \varepsilon$ time units. Hence, LAZY(α) takes at least

$$(\alpha(2 + 3\alpha)) + (3 + 3\alpha - 2\varepsilon) + (2 + 2\alpha - \varepsilon) = 5 + 7\alpha + 3\alpha^2 - 3\varepsilon$$

time units to serve all requests.

Case 2 ($\alpha \in [\frac{2}{3}, \frac{\sqrt{37-12\varepsilon}-1}{6}]$): We define the instance by giving the requests

$$\begin{aligned}
r_1 &= (0, 1; \varepsilon),\\
r_2 &= (-\alpha, -\alpha; \alpha + \varepsilon),\\
r_3 &= \Big(\frac{2}{\alpha} + 1 - 2\alpha - \varepsilon, \frac{2}{\alpha} + 1 - 2\alpha - \varepsilon; \alpha + 2\varepsilon\Big),\\
r_4 &= (2 + \alpha - \varepsilon, 2 + \alpha - \varepsilon; 2 + \alpha - \varepsilon),\\
r_5 &= (2 + \alpha - \varepsilon, 2 + \alpha - \varepsilon; 2 + 3\alpha).
\end{aligned}$$

One solution is to move to $-\alpha$ and wait there until time $\alpha + \varepsilon$. Then, r_2 can be served, and the server can move to 0 where it arrives at time $2\alpha+\varepsilon$. It picks up r_1 and delivers it at time $2\alpha+1+\varepsilon$ at 1. It keeps moving to position $\frac{2}{\alpha}+1-2\alpha-\varepsilon$, where it arrives at time $\frac{2}{\alpha} + 1$ and immediately serves r_3. It continues to move to $2 + \alpha - \varepsilon$ and serves r_4 and r_5 there at time $2 + 3\alpha$. Since the last request is released at time $2 + 3\alpha$, we have

$$\textsc{Opt} = \textsc{Opt}(2 + 3\alpha) = 2 + 3\alpha.$$

We now analyze what $\textsc{Lazy}(\alpha)$ does. We have $\textsc{Opt}(\varepsilon) = 1 + \varepsilon$. Thus, the server starts waiting in 0 until time $\alpha(1 + \varepsilon)$. At time $\alpha(1 + \varepsilon)$, the server starts an optimal schedule over all unserved requests, i.e., over $\{r_1\}$. It picks up r_1 and starts moving towards 1. At time $\alpha + \varepsilon > \alpha(1 + \varepsilon)$, request r_2 arrives. We have $\textsc{Opt}(\alpha + \varepsilon) = 2\alpha + 1 + \varepsilon$. Serving the loaded request r_1 and returning to the origin would take the server until time

$$\alpha(1 + \varepsilon) + 2 > \alpha(2\alpha + 1 + \varepsilon) = \alpha\textsc{Opt}(\alpha + \varepsilon),$$

i.e., the server continues its current schedule. At time $\alpha + 2\varepsilon$, r_3 is released. We have $\textsc{Opt}(\alpha + 2\varepsilon) = \frac{2}{\alpha} + 1$. Serving the loaded request and returning to the origin would still take until time

$$\alpha(1 + \varepsilon) + 2 > 2 + \alpha = \alpha\textsc{Opt}(\alpha + 2\varepsilon).$$

Thus, the server continues the current schedule which is finished at time $1 + \alpha$ in position 1. Since $1 + \alpha < 2 + \alpha = \alpha\textsc{Opt}(\alpha + 2\varepsilon)$, the server starts waiting in 1 until time $2 + \alpha$. At time $2 + \alpha - \varepsilon$, request r_4 is released. We have $\textsc{Opt}(2 + \alpha - \varepsilon) = 2 + 3\alpha$. It would take the server until time

$$3 + \alpha - \varepsilon > 2\alpha + 3\alpha^2 = \alpha\textsc{Opt}(2 + \alpha - \varepsilon)$$

to return to the origin, where the inequality follows from the fact that $\alpha < \frac{\sqrt{37-12\varepsilon}-1}{6}$. Thus, the server starts waiting until time $\alpha\textsc{Opt}(2 + \alpha - \varepsilon) = 2\alpha + 3\alpha^2$. After it finished waiting, it starts the second schedule and tries to serve r_3, r_4 and then r_2. At time $2 + 3\alpha$, request r_5 is released. Since this does not change the completion time of the optimum and because the server started the current schedule at time $\alpha\textsc{Opt}(\alpha+2\varepsilon)$, it continues its current schedule. The second schedule takes $(1 + \alpha - \varepsilon) + (2 + 2\alpha - \varepsilon) = 3 + 3\alpha - 2\varepsilon$

time units and ends in $-\alpha$. The last schedule, in which r_5 is served, is started immediately and takes $2+2\alpha-\varepsilon$ time units. Hence, LAZY(α) takes until time

$$(\alpha(2+3\alpha))+(3+3\alpha-2\varepsilon)+(2+2\alpha-\varepsilon)=5+7\alpha+3\alpha^2-3\varepsilon$$

to serve all requests.

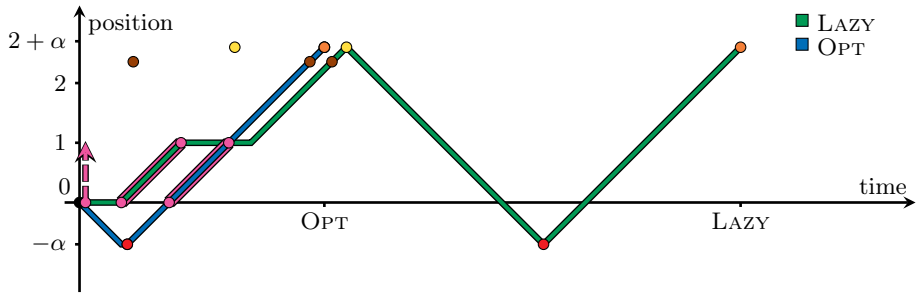

Fig. 4. Instance of the open online dial-a-ride problem on the line where LAZY(α) has a competitive ratio of at least $2+\alpha\frac{1-\alpha}{2+3\alpha}$ for all $\alpha\in[0,1)$. This is the construction of Case 2 in the proof of Theorem 3.

Case 3 ($\alpha\in[\frac{\sqrt{37-12\varepsilon}-1}{6},1)$): We define the instance by giving the requests

$$\begin{aligned}
r_1&=(0,1;\varepsilon),\\
r_2&=(-\alpha,-\alpha;\alpha+\varepsilon),\\
r_3&=\Big(\frac{2}{\alpha}+1-2\alpha-\varepsilon,\frac{2}{\alpha}+1-2\alpha-\varepsilon;\alpha+2\varepsilon\Big),\\
r_4&=\Big(\frac{3}{\alpha}+1-2\alpha-\frac{(2+\alpha)\varepsilon}{\alpha},\frac{3}{\alpha}+1-2\alpha-\frac{(2+\alpha)\varepsilon}{\alpha};2+\alpha-\varepsilon\Big),\\
r_5&=(2+\alpha-\varepsilon,2+\alpha-\varepsilon;3+\alpha-3\varepsilon),\\
r_6&=(2+\alpha-\varepsilon,2+\alpha-\varepsilon;2+3\alpha).
\end{aligned}$$

One solution is to move to $-\alpha$ and wait there until time $\alpha+\varepsilon$. Then, r_2 can be served, and the server can move to 0 where it arrives at time $2\alpha+\varepsilon$. It picks up r_1 and delivers it at time $2\alpha+1+\varepsilon$ at 1. It keeps moving to position $\frac{2}{\alpha}+1-2\alpha-\varepsilon$, where it arrives at time $\frac{2}{\alpha}+1$ and immediately serves r_3. It continues to move to $\frac{3}{\alpha}+1-2\alpha-\frac{(2+\alpha)\varepsilon}{\alpha}$, where it arrives at time $\frac{3}{\alpha}+1-\frac{2\varepsilon}{\alpha}$ and immediately serves r_4. Lastly, it moves to $2+\alpha-\varepsilon$, where it arrives at time $2+3\alpha$ and serves r_4 and r_5 there. Since the last request is released at time $2+3\alpha$, we have

$$\text{OPT}=\text{OPT}(2+3\alpha)=2+3\alpha.$$

We now analyze what LAZY(α) does. We have $\text{OPT}(\varepsilon)=1+\varepsilon$. Thus, the server starts waiting in 0 until time $\alpha(1+\varepsilon)$. At time $\alpha(1+\varepsilon)$, the server starts an

optimal schedule over all unserved requests, i.e., over $\{r_1\}$. It picks up r_1 and starts moving towards 1. At time $\alpha+\varepsilon > \alpha(1+\varepsilon)$, request r_2 arrives. We have $\textsc{Opt}(\alpha+\varepsilon) = 2\alpha+1+\varepsilon$. Serving the loaded request r_1 and returning to the origin would take the server until time

$$\alpha(1+\varepsilon)+2 > \alpha(2\alpha+1+\varepsilon) = \alpha\textsc{Opt}(\alpha+\varepsilon),$$

i.e., the server continues its current schedule. At time $\alpha+2\varepsilon$, request r_3 is released. Now, we have $\textsc{Opt}(\alpha+2\varepsilon) = \frac{2}{\alpha}+1$. Serving the loaded request and returning to the origin would still take until time

$$\alpha(1+\varepsilon)+2 > 2+\alpha = \alpha\textsc{Opt}(\alpha+2\varepsilon).$$

Thus, the server continues the current schedule which is finished at time $1+\alpha$ in position 1. Since $1+\alpha < 2+\alpha = \alpha\textsc{Opt}(\alpha+2\varepsilon)$, the server starts waiting in 1 until time $2+\alpha$. At time $2+\alpha-\varepsilon$, request r_4 is released. We have $\textsc{Opt}(2+\alpha-\varepsilon) = \frac{3}{\alpha}+1-\frac{2\varepsilon}{\alpha}$. It would take the server until time

$$3+\alpha-\varepsilon > 3+\alpha-2\varepsilon = \alpha\textsc{Opt}(2+\alpha-\varepsilon),$$

to return to the origin, i.e., the server starts waiting until time $\alpha\textsc{Opt}(2+\alpha-\varepsilon) = 3+\alpha-2\varepsilon$. At time $3+\alpha-3\varepsilon$, request r_5 is released. We have $\textsc{Opt}(3+\alpha-3\varepsilon) = 2+3\alpha$. It would take the server until time

$$4+\alpha-3\varepsilon > 2\alpha+3\alpha^2 = \alpha\textsc{Opt}(3+\alpha-3\varepsilon)$$

to return to the origin, where the inequality follows from the fact that $\alpha < 1$ and that ε is small. Thus, the server waits in 1 until time $\alpha\textsc{Opt}(3+\alpha-3\varepsilon) = 2\alpha+3\alpha^2$ and then starts its second schedule to serve requests r_3, r_4, r_5 and then r_2. At time $2+3\alpha$, request r_6 is released. Since this does not change the completion time of the optimum and because the server started the current schedule at time $\alpha\textsc{Opt}(3+\alpha-3\varepsilon)$, it continues its current schedule. The second schedule takes $(1+\alpha-\varepsilon)+(2+2\alpha-\varepsilon) = 3+3\alpha-2\varepsilon$ time units and ends in $-\alpha$. The last schedule, in which r_6 is served, is started immediately and takes $2+2\alpha-\varepsilon$ time units. Hence, $\textsc{Lazy}(\alpha)$ takes until time

$$(\alpha(2+3\alpha))+(3+3\alpha-2\varepsilon)+(2+2\alpha-\varepsilon) = 5+7\alpha+3\alpha^2-3\varepsilon$$

to serve all requests.

In all three cases, the optimal solution is $2+3\alpha$ and the algorithm takes at least $5+7\alpha+3\alpha^2-3\varepsilon$ time units. Thus, the competitive ratio of $\textsc{Lazy}(\alpha)$ is at least

$$\frac{5+7\alpha+3\alpha^2-3\varepsilon}{2+3\alpha} = 2+\alpha+\frac{1-\alpha-3\varepsilon}{2+3\alpha}.$$

The statement follows by taking the limit $\varepsilon \to 0$.

Now, we combine our results for the lower bounds (cf. Figure 5). Combining Lemma 1 and Proposition 1, we obtain that, for $\alpha \geq 1$, $\textsc{Lazy}(\alpha)$ has a

competitive ratio of at least $\max\{1+\alpha, 2+2/3\alpha\}$, which proves the lower bound of Theorem 1. Minimizing over $\alpha \geq 1$ yields a competitive ratio of at least $3/2+\sqrt{11/12} > 2.457$ in that domain. For the case $\alpha < 1$, we have seen in Proposition 2 that the algorithm Lazy(α) has a competitive ratio of at least $1+3/(\alpha+1) > 5/2$. Together, this proves Corollary 1.

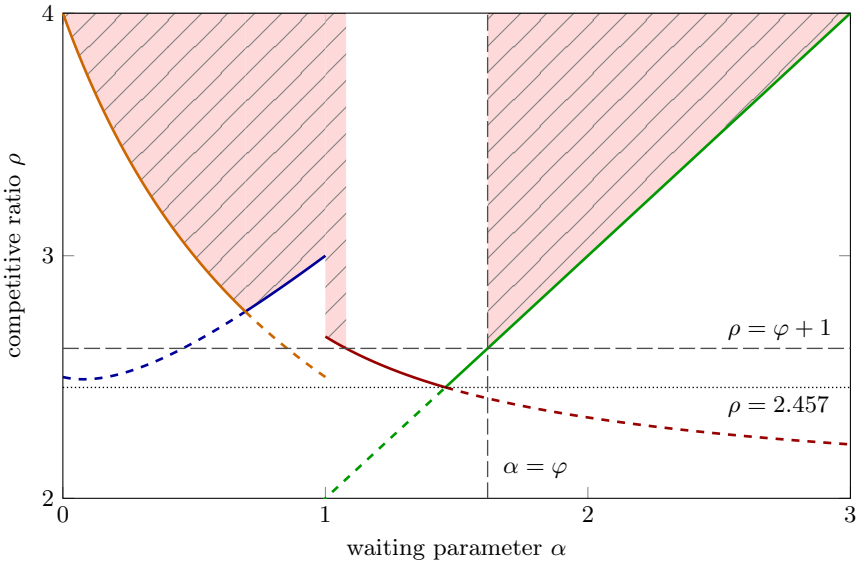

Fig. 5. Lower bounds on the competitive ratio of Lazy(α) depending on α. The lower bound of Lemma 1 is depicted in green, the lower bound of Proposition 1 in red, the lower bound of Proposition 2 in orange, and the lower bound of Proposition 3 in blue. The highlighted area over the plot indicates the domain of α for which no improvement over $1+\varphi$ is possible.

Moreover, we conclude that the results above narrow the range for α in which Lazy(α) might have a competitive ratio better than $\varphi+1$. By Lemma 1, it follows that Lazy(α) cannot have a better competitive ratio than $\varphi+1$ for any $\alpha > \varphi \approx 1.618$. By Proposition 1, we obtain that Lazy(α) has competitive ratio at least $\varphi+1$ for any α with $1 \leq \alpha \leq \frac{2\varphi}{3} \approx 1.079$. Proposition 2 yields that, for $0 \leq \alpha \leq 0.695$, the competitive ratio of Lazy(α) is at least $2.768 > \varphi+1$. Lastly, for $0.695 < \alpha < 1$, Proposition 3 gives a lower bound of 2.768 on the competitive ratio of Lazy(α). To summarize, an improvement of the competitive ratio of Lazy(α) might only be possible for some $\alpha \in (2\varphi/3, \varphi) \approx [1.08, 1.618)$, which proves Corollary 2.

References

1. Allulli, L., Ausiello, G., Laura, L.: On the power of lookahead in on-line vehicle routing problems. In: Proceedings of the 11th Annual International Computing and Combinatorics Conference (COCOON). pp. 728–736 (2005)
2. Ascheuer, N., Krumke, S.O., Rambau, J.: Online dial-a-ride problems: Minimizing the completion time. In: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS). pp. 639–650 (2000)
3. Ausiello, G., Demange, M., Laura, L., Paschos, V.: Algorithms for the on-line quota traveling salesman problem. Information Processing Letters **92**(2), 89–94 (2004)
4. Ausiello, G., Feuerstein, E., Leonardi, S., Stougie, L., Talamo, M.: Algorithms for the on-line travelling salesman. Algorithmica **29**(4), 560–581 (2001)
5. Ausiello, G., Allulli, L., Bonifaci, V., Laura, L.: On-line algorithms, real time, the virtue of laziness, and the power of clairvoyance. In: Proceddings of the 3rd International Conference on Theory and Applications of Models of Computation (TAMC). pp. 1–20 (2006)
6. Bienkowski, M., Kraska, A., Liu, H.: Traveling repairperson, unrelated machines, and other stories about average completion times. In: Bansal, N., Merelli, E., Worrell, J. (eds.) Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP). pp. 28:1–28:20 (2021)
7. Bienkowski, M., Liu, H.: An improved online algorithm for the traveling repairperson problem on a line. In: Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS). pp. 6:1–6:12 (2019)
8. Birx, A.: Competitive analysis of the online dial-a-ride problem. Ph.D. thesis, TU Darmstadt (2020)
9. Birx, A., Disser, Y.: Tight analysis of the smartstart algorithm for online dial-a-ride on the line. SIAM Journal on Discrete Mathematics **34**(2), 1409–1443 (2020)
10. Birx, A., Disser, Y., Schewior, K.: Improved bounds for open online dial-a-ride on the line. In: Proceedings of the 22nd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX). vol. 145, p. 21(22) (2019)
11. Bjelde, A., Disser, Y., Hackfeld, J., Hansknecht, C., Lipmann, M., Meißner, J., Schewior, K., Schlöter, M., Stougie, L.: Tight bounds for online tsp on the line. ACM Transactions on Algorithms **17**(1) (2020)
12. Blom, M., Krumke, S.O., de Paepe, W.E., Stougie, L.: The online TSP against fair adversaries. INFORMS Journal on Computing **13**(2), 138–148 (2001)
13. Bonifaci, V., Stougie, L.: Online k-server routing problems. Theory of Computing Systems **45**(3), 470–485 (2008)
14. Feuerstein, E., Stougie, L.: On-line single-server dial-a-ride problems. Theoretical Computer Science **268**(1), 91–105 (2001)
15. Hauptmeier, D., Krumke, S., Rambau, J., Wirth, H.C.: Euler is standing in line dial-a-ride problems with precedence-constraints. Discrete Applied Mathematics **113**(1), 87–107 (2001)
16. Hauptmeier, D., Krumke, S.O., Rambau, J.: The online dial-a-ride problem under reasonable load. In: Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC). pp. 125–136 (2000)
17. Jaillet, P., Lu, X.: Online traveling salesman problems with service flexibility. Networks **58**(2), 137–146 (2011)
18. Jaillet, P., Lu, X.: Online traveling salesman problems with rejection options. Networks **64**(2), 84–95 (2014)

19. Jaillet, P., Wagner, M.R.: Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses. Operations Research **56**(3), 745–757 (2008)
20. Jawgal, V.A., Muralidhara, V.N., Srinivasan, P.S.: Online travelling salesman problem on a circle. In: In Proceedings of the 15th International Conference on Theory and Applications of Models of Computation (TAMC). pp. 325–336 (2019)
21. Krumke, S.O.: Online optimization competitive analysis and beyond. Habilitation thesis, Zuse Institute Berlin (2001)
22. Krumke, S.O., Laura, L., Lipmann, M., Marchetti-Spaccamela, A., de Paepe, W., Poensgen, D., Stougie, L.: Non-abusiveness helps: An O(1)-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX). pp. 200–214 (2002)
23. Krumke, S.O., de Paepe, W.E., Poensgen, D., Lipmann, M., Marchetti-Spaccamela, A., Stougie, L.: On minimizing the maximum flow time in the online dial-a-ride problem. In: Proceedings of the 3rd International Conference on Approximation and Online Algorithms (WAOA). pp. 258–269 (2006)
24. Krumke, S.O., de Paepe, W.E., Poensgen, D., Stougie, L.: News from the online traveling repairman. Theoretical Computer Science **295**(1-3), 279–294 (2003)
25. Lipmann, M., Lu, X., de Paepe, W.E., Sitters, R.A., Stougie, L.: On-line dial-a-ride problems under a restricted information model. Algorithmica **40**(4), 319–329 (2004)
26. Lippmann, M.: On-line routing. Ph.D. thesis, Technische Universiteit Eindhoven (2003)
27. Yi, F., Tian, L.: On the online dial-a-ride problem with time-windows. In: Proceedings of the 1st International Conference on Algorithmic Applications in Management (AAIM). pp. 85–94 (2005)