Tight Bounds for Online TSP on the Line
Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Kevin Schewior, Miriam Schlöter, Leen Stougie

To cite this version:
Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, et al.. Tight Bounds for Online TSP on the Line. ACM-SIAM Symposium on Discrete Algorithms (SODA), Jan 2017, Barcelona, Spain. pp.994 - 1005, 10.1137/1.9781611974782.63. hal-01549685

HAL Id: hal-01549685
https://hal.inria.fr/hal-01549685
Submitted on 29 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Tight Bounds for Online TSP on the Line

Antje Bjelde ∗ † Yann Disser‡ Jan Hackfeld∗§ Christoph Hansknecht¶
Maarten Lipmann∥ Julie Meißner∗ ** Kevin Schewior† †‡‡ Miriam Schlöter∗§
Leen Stougie §§

Abstract
We consider the online traveling salesperson problem (TSP), where requests appear online over time on the real line and need to be visited by a server initially located at the origin. We distinguish between closed and open online TSP, depending on whether the server eventually needs to return to the origin or not. While online TSP on the line is a very natural online problem that was introduced more than two decades ago, no tight competitive analysis was known to date. We settle this problem by providing tight bounds on the competitive ratios for both the closed and the open variant of the problem. In particular, for closed online TSP, we provide a 1.64-competitive algorithm, thus matching a known lower bound. For open online TSP, we give a new upper bound as well as a matching lower bound that establish the remarkable competitive ratio of 2.04.

Additionally, we consider the online Dial-A-Ride problem on the line, where each request needs to be transported to a specified destination. We provide an improved non-preemptive lower bound of 1.75 for this setting, as well as an improved preemptive algorithm with competitive ratio 2.41.

Finally, we generalize known and give new complexity results for the underlying offline problems. In particular, we give an algorithm with running time $O(n^2)$ for closed offline TSP on the line with release dates and show that both variants of offline Dial-A-Ride on the line are NP-hard for any capacity $c \geq 2$ of the server.

1 Introduction
In the online Traveling Salesperson Problem (TSP) on the line, we consider a server initially located at the origin of the real line that has to serve requests that appear over time. The server has unit speed and serves requests (in any order) by moving to the position of the corresponding request at some time after its release. The objective in online TSP on the line is to minimize the makespan, i.e., the time until all requests have been served. In the closed variant of the problem, the server needs to return to the origin after serving all requests, while the open variant has no such requirement.

Online TSP is a natural online problem similar to the classical k-server problem [20]. In the latter, the order in which requests need to be served is prescribed, and the problem thus becomes trivial on the line for $k = 1$ server. In contrast, online TSP on the line is a non-trivial problem that arises in 1-dimensional collection/delivery problems. Examples include robotic welding/screwing/depositing material, horizontal/vertical item delivery systems, and the collection of objects from mass storage shelves. The online Dial-A-Ride problem additionally allows transportation requests that specify a source and destination that need to be visited by the server in this order. If the capacity of the server is finite, it limits the number of requests that can be transported simultaneously. The online Dial-A-Ride problem on the line arises, e.g., when controlling industrial or personal elevators.

While both online TSP and online Dial-A-Ride on the line are among the most natural online problems and have been studied extensively over the last two decades [2, 4, 5, 6, 7, 10, 14, 15, 16, 17], no satisfactory (tight) analysis was known for either problem in terms of competitive ratios. We address
this shortcoming for TSP on the line by providing a tight upper bound for the closed variant, as well as tight bounds for the open variant. We emphasize that our results for the open and closed variant of the problem are independent and require substantially different approaches. Aside from our results for online TSP, we narrow the gaps for online Dial-A-Ride on the line by giving improved bounds. In addition to online results, we study the computational complexity of the underlying offline problems.

1.1 Our results

We have the following results (cf. Tables 1 and 2):

**Tight bounds for online TSP on the line.** Our main results are best-possible online algorithms for both the open and closed variant of online TSP on the line, as well as a new (tight) lower bound for the open variant. Our algorithm for the closed variant has a competitive ratio of \((9 + \sqrt{17})/8 \approx 1.64\), matching a lower bound of Ausiello et al. [6] and improving on their 1.75-competitive algorithm. For open TSP on the line, we give a lower bound of 2.04 on the competitive ratio, which is the first bound strictly greater than 2. We also provide an optimal online algorithm matching this bound and improving on the 2.33-competitive algorithm by Ausiello et al. [6]. Our results settle online TSP on the line from the perspective of competitive analysis.

**Improved bounds for online Dial-A-Ride on the line.** Our lower bounds for online TSP on the line immediately apply to preemptive and non-preemptive online Dial-A-Ride on the line. In particular, our lower bound of 2.04 is the first bound greater than 2 for the open variant of the problem. Additionally, we provide a simple preemptive 2.41-competitive algorithm, which improves a (non-preemptive) 3.41-competitive algorithm by Krumke [15]. For the closed Dial-A-Ride variant, the lower bound of 1.64 by Ausiello et al. [6] was improved for one server with unit capacity without preemption to 1.71 by Ascheuer et al. [2]. We improve this bound further to 1.75 for any finite capacity \(c \geq 1\). The best known algorithm for closed Dial-A-Ride on the line for finite capacity \(c \geq 1\) is 2-competitive and was given by Ascheuer et al. [2].

**New offline complexity results.** Regarding offline TSP on the line with release times, Psaraftis et al. [21] showed a dynamic program that solves the open variant in quadratic time. We refute their claim that all optimal closed tours have a very simple structure with a counterexample, and we adapt their algorithm to find an optimal closed tour in quadratic time. For the non-preemptive offline Dial-A-Ride problem on the line, results have previously been obtained for the closed variant without release times. For capacity \(c = 1\) Gilmore and Gomory [12] and Atallah and Kosaraju [3] gave polynomial time algorithms, and Guan [13] proved hardness for the case \(c = 2\). We show that both the open and closed variant of the problem are NP-hard for any capacity \(c \geq 2\). Additionally, we show that the case with release times and any \(c \geq 1\) is NP-hard. The complexity of offline Dial-A-Ride on the line with unbounded capacity remains open.

1.2 Further related work

For the online TSP problem in general metric spaces, Ausiello et al. [6] show a lower bound of 2 on the competitive ratio for the open version and a 1.64 lower bound for the closed version, both bounds being achieved on the real line. For the open online TSP, they present a 2.5-competitive algorithm, and for the closed version they give a 2-competitive algorithm. Jaillet and Wagner [14] give 2-competitive algorithms for the closed version that can additionally deal with precedence constraints or multiple servers. Blom et al. [7] consider the closed online TSP problem on the non-negative part of the real line and present a best possible algorithm with competitive ratio 1.5. They also study a “fair” setting where the optimum does not travel outside the convex hull of the known requests, and they derive an algorithm for the real half-line with a better competitive ratio of 1.28 for this setting. Krumke et al. [17] show that there cannot be a competitive algorithm for open online TSP with the objective of minimizing the maximum flow time instead of minimizing the makespan. For the real line they define a fair setting and give a competitive algorithm for it.

The **online repairperson problem** is the open online TSP problem with the objective of minimizing the weighted sum of completion times. Feuerstein and Stougie [10] show a lower bound of 5.83 on the best-possible competitive ratio for this problem and provide a 9-competitive algorithm for the real line. Krumke et al. [16] give a best-possible online algorithm with competitive ratio 5.83 for general metric spaces.

For the closed online Dial-A-Ride problem without preemption, Feuerstein and Stougie [10] show a lower bound of 2 for the competitive ratio in general, and present an algorithm with a best-possible competitive ratio of 2 for the case that the server has infinite capacity. Ascheuer et al. [2] analyze
different algorithms for the same setting and present a 2-competitive algorithm for any finite capacity \( c \geq 1 \).

For minimizing the sum of completion times instead of the makespan, Feuerstein and Stougie [10] further show a lower bound of 3 for a server with unit capacity and a lower bound of 2.41 independent of the capacity. Moreover, they provide a 15-competitive algorithm for the real line and unlimited capacity. For the same objective function, Krumke et al. [16] present an algorithm with a competitive ratio of 5.83 for a server with unit capacity in an arbitrary metric space.

The offline version of the TSP problem is a well-studied NP-hard problem (e.g., see [18]). Afrati et al. [1] show that the offline traveling repairperson problem is NP-hard in general, but can be solved in time \( O(n^2) \) for the real line and unweighted sum of completion times objective. There are many offline variants of the DIAL-A-RIDE problem, differing in capacities, the underlying metric space, release times and deadlines, open versus closed tours, and in whether preemption is allowed (e.g., see [9]). The special case without release times and unit capacity is known as the stacker crane problem. Attallah and Kosaraju [3] present a polynomial algorithm for the closed, non-preemptive stacker crane problem on the real line. Frederickson and Guan [11] show that this problem is NP-complete on trees. Guan [13] shows that the DIAL-A-RIDE problem remains easy on the line with capacities larger than one if preemption is allowed, and that it remains hard on trees. Finally, Charikar and Raghavachari [8] give a \( O(\sqrt{c} \log n \log \log n) \)-approximation for the closed DIAL-A-RIDE problem in metric spaces with \( n \) points and without preemption. In the same paper they claim a 2-approximation for the problem on the line, however this result seems to be incorrect (personal communication).

2 Problem definition and notation

We consider a server that moves along the real line with (at most) unit speed. We let \( p_i \) denote the position of the server at time \( t \geq 0 \) and assume (without loss of generality) that \( p_0 = 0 \). With this notation, the speed limitation of the server can equivalently be expressed via \( |p_i - p_{i'}| \leq |t - t'| \) for all \( t, t' \geq 0 \). A series of requests \( \sigma_1, \ldots, \sigma_n \) arrives over time with \( \sigma_i = (a_i, b_i; t_i) \), where \( t_i \geq 0 \) denotes the release time of the request and \( a_i, b_i \in \mathbb{R} \) denote its source and target position, respectively. For TSP, we have \( a_i = b_i \) and write \( \sigma_i = (a_i; t_i) \). If not stated otherwise, we assume \( t_1 \leq t_2 \leq \ldots \leq t_n \). Moreover, we assume without loss of generality that \( t_i \geq |a_i| \) holds because the server can not reach \( \sigma_i \) before time \( |a_i| \) and it only helps the algorithm to know a request earlier. We further use the notation \( A^R := \max_{i=1, \ldots, n}\{a_i, b_i\} \) to denote the rightmost point that needs to be visited by the server, and similarly \( A^L := \min_{i=1, \ldots, n}\{a_i, b_i\} \). Here and throughout we refer to the negative direction of the real line as left and the positive direction as right.

In both TSP and DIAL-A-RIDE on the line, all requests need to be served. For TSP, we consider a request served if \( p_t = a_i \) for some time \( t \geq t_i \). For DIAL-A-RIDE, the server may collect request \( \sigma_i \) at time \( t \geq t_i \) if \( p_t = a_i \). In the preemptive DIAL-A-RIDE problem, the server can drop off any request.
it is carrying at its current location at any time. If request \( \sigma_i \) is dropped off at point \( p \) at time \( t \), we consider it to be modified to the new request \((p,b; t)\). In the non-preemptive DIAL-A-RIDE problem, the server may only drop off a request at its target location. We consider a request served if it is ever dropped off at its target location.

In TSP on the line, the behavior of the server in our algorithms at time \( t \) will mostly depend on so-called extreme requests. For \( t \geq 0 \), we denote by \( \sigma^R(t) = (a^R(t); t^R(t)) \) the unserved request that is rightmost of the position of the server \( p_t \), provided such a request exists, i.e., the unserved request \( \sigma = (a;t') \) with \( t' \leq t \), \( a > p_t \), and maximizing \( a \). Analogously, \( \sigma^L(t) = (a^L(t); t^L(t)) \) denotes unserved request that is leftmost of the position of the server \( p_t \). If there is more than one right-most (left-most) request, we choose the one with the largest release time.

If the server has finite capacity \( c \geq 1 \), it can carry at most \( c \) requests at any time. We assume that no time is needed for picking up and dropping off requests, so that the server can pick up and drop off any number of requests at the same time, as long as its capacity is not exceeded.

We refer to a valid trajectory of the server together with the description of when it picks up and drops requests as a tour \( T \). If the tour ends at \( p_0 = 0 \), we call it closed, otherwise it is open. We denote the makespan of the tour \( T \) by \(|T|\). The objective in the open (closed) version of both TSP and DIAL-A-RIDE is to find an open (closed) tour \( T \) that serves all requests and minimizes \(|T|\).

In the offline setting, we assume all requests to be known from the start. We let \( T^{\text{Opt}} \) denote an optimal offline tour. In the online setting, we assume that request \( \sigma_i \) is revealed at its release time \( t_i \), at which point the tour of the server until time \( t_i \) must already have been fixed irrevocably. Additionally, we assume that the total number \( n \) of requests is unknown. We measure the quality of an online algorithm via its competitive ratio, i.e., the maximum over all sequences of requests of the ratio of the makespan of the tour it produces and \(|T^{\text{Opt}}|\).

In order to describe the trajectory of the server, we use the notation “move(a)” for the tour that moves the server from its current position with unit speed to the point \( a \in \mathbb{R} \) and the notation “waituntil(s)” for the tour that keeps the server stationary until time \( s \). We use the operator \( \oplus \) to concatenate tours. For example, if \( T_0 \) is a tour of the server that ends at time \( t_0 \) at position \( p_{t_0} \), then \( T_0 \oplus \text{move}(a) \) describes the tour that ends at time \( t_0 + |a - p_{t_0}| \), is identical to the tour \( T_0 \) until time \( t_0 \) and satisfies \( p_t = p_{t_0} + (a - p_{t_0})(t-t_0) \) for \( t_0 \leq t \leq t_0 + |a - p_{t_0}| \). Similarly, \( T_0 \oplus \text{waituntil}(s) \) is the tour that ends at time \( \max\{t_0, s\} \) identical to the tour \( T_0 \) until time \( t_0 \) and that satisfies \( p_{t_0} = p_t \) for all \( s \in [t_0, s] \).

We skip the proofs of our results as they are very technical. They can be found in the full version of the paper.

### 3 Algorithm for closed online TSP

In this section we consider the closed online TSP problem and describe a best-possible algorithm with competitive ratio \( \rho = (9 + \sqrt{17})/8 \approx 1.64 \), where \( \rho \) is the nonnegative root of the polynomial \( 4x^2 - 9x + 4 \).

We start by developing some intuition for our algorithm. In the following \( T^{\text{Alg}} \) is the tour derived by an algorithm Alg. Observe that the decision of how to move the server at time \( t \) only depends on its position \( p_t \) and the location of the left- and rightmost extreme requests \( \sigma^L(t) = (a^L(t); t^L(t)) \) and \( \sigma^R(t) = (a^R(t); t^R(t)) \). All other requests can be served during any tour serving \( \sigma^L(t) \) and \( \sigma^R(t) \). We will show that in this setting we can assume that \( a^L(t) \leq 0 \) and \( a^R(t) \geq 0 \), provided these extremes exist. If \( \sigma^R(t) \) (resp. \( \sigma^L(t) \)) does not exist we set \( a^R(t) = t^R(t) = 0 \) (resp. \( a^L(t) = t^L(t) = 0 \)). Thus, in contrast to the initial definition of extreme requests, in our setting a leftmost extreme is always left and a rightmost extreme is always right of the origin. When both extremes exist, we have, on a high level, three possible courses of action at time \( t \). Either we immediately decide to serve \( \sigma^L(t) \) and \( \sigma^R(t) \) in one of the two possible orders, or we wait for some time for additional information to make a more informed decision. Intuitively, the critical case for our competitiveness is the case where we decide to serve \( \sigma^L(t) \) and \( \sigma^R(t) \) in a different order than \( T^{\text{Opt}} \). Let \( T_{RL}(t) \) and \( T_{LR}(t) \) be the tours that start at the origin at time \( 0 \) and then move as follows,

\[
T_{RL}(t) = \text{waituntil}(t^R(t) - |a^R(t)|) \oplus \text{move}(a^R(t)) \\
\quad \oplus \text{move}(a^L(t)) \oplus \text{move}(p_0),
\]

\[
T_{LR}(t) = \text{waituntil}(t^L(t) - |a^L(t)|) \oplus \text{move}(a^L(t)) \\
\quad \oplus \text{move}(a^R(t)) \oplus \text{move}(p_0).
\]
tour $T_{RL}(t)$, but continues to move to the left after serving $\sigma^L(t)$. The time when $T^{Opt}$ reaches $a^L(t)$ is $t' = t^R(t) + |a^R(t)| + |a^L(t)|$, since $t^R(t) \geq |a^R(t)|$ by assumption. Let $t_0$ be the time when we reach the origin $p_0$ after serving $\sigma^L(t)$, and assume that $t' \leq t_0$. Now a new request $\sigma' = (p'; t_0)$ may arrive at time $t_0$ and position $p' = -|a^L(t)| - (t_0 - t') = -t_0 + t^R(t) + |a^L(t)|$, that the optimum can serve immediately at time $t_0$. We then have

$$|T^{Opt}| = t_0 + |p'| = t_0 + |a^L(t)| + (t_0 - t') = 2t_0 - t^R(t) - |a^R(t)|.$$ 

Our algorithm still needs to serve $\sigma'$ and $\sigma^R$ at time $t_0$, and hence

$$|T^{Alg}| = t_0 + 2|p'| + 2|a^R(t)| = 3t_0 - 2t^R(t).$$

For the algorithm to be $\rho$-competitive, we need $|T^{Alg}|/|T^{Opt}| \leq \rho$, and we thus obtain a condition on the earliest time $t_0$ we may return to the origin.

**FACT 3.1.** If at time $t$ a $\rho$-competitive algorithm serves the leftmost extreme $\sigma^L(t)$ first and $t_0$ denotes the first time the server returns to the origin after having served $\sigma^L(t)$, then

$$t_0 \geq t_0^L(t) := \frac{\rho|a^R(t)| - (2 - \rho)t^R(t)}{2\rho - 3}.$$ 

A symmetric statement with $t_0^R(t)$ holds if the algorithm serves $\sigma^R(t)$ first.

Fact 3.1 illustrates that waiting is sometimes necessary in order to be competitive. On the other hand, we can obviously not afford to wait too long. To quantify this, we introduce a lower bound on the length of $T^{Opt}$.

**DEFINITION 3.1.** If both extreme request exist at time $t$ we define the greedy tour $T^{greedy}(t)$ at time $t$ as

$$T^{greedy}(t) := \begin{cases} T_{LR}(t), & \text{if } |T_{LR}(t)| \leq |T_{RL}(t)|, \\ T_{RL}(t), & \text{else.} \end{cases}$$

If only $\sigma^L(t)$ exists, we set $T^{greedy}(t) := T_{LR}(t)$ and we set $T^{greedy}(t) := T_{RL}(t)$ if only $\sigma^R(t)$ exists.

**OBSERVATION 3.1.** We have

$$|T_{RL}(t)| = t^R(t) + |a^R(t)| + 2|a^L(t)|, \\
|T_{LR}(t)| = t^L(t) + |a^L(t)| + 2|a^R(t)|$$

and $|T^{greedy}(t)| = \min\{|T_{LR}(t)|, |T_{RL}(t)|\} \leq |T^{Opt}|$ if both extremes exist at time $t$.

Assume we are still waiting at the origin at time $t$, i.e., $p_t = 0$. From Observation 3.1, we conclude that if $t \leq \rho|T^{greedy}(t)| - 2|a^R(t)| - 2|a^L(t)|$, we can wait until time $t$ $|T^{greedy}(t)| - 2|a^R(t)| - 2|a^L(t)|$, and then still serve $\sigma^R(t)$ and $\sigma^L(t)$ and return to the origin $p_0$ until time $t$ $|T^{greedy}| \leq |T^{Opt}|$, i.e., we can stay $\rho$-competitive. Formally, we make the following definition.

**DEFINITION 3.2.** Let the safe tour $T^{safe}(t)$ at time $t$ be defined as

$$T^{safe}(t) := \begin{cases} T^{wait} \oplus T_{LR}(t), & \text{if } |a^L(t)| \geq |a^R(t)|, \\ T^{wait} \oplus T_{RL}(t), & \text{else}, \end{cases}$$

with

$$T^{wait} := \text{waituntil}(\rho|T^{greedy}(t)| - 2|a^R(t)| - 2|a^L(t)|).$$

We still have to ensure that the definition of $T^{safe}(t)$ is compatible with the requirement from Fact 3.1 regarding the time $t_0$ when the tour first returns to the origin. In case $|a^L(t)| \geq |a^R(t)|$, we get $t_0 \geq t_0^L(t)$ with $t_0 = \rho|T^{greedy}(t)| - 2|a^R(t)| \geq (4\rho - 2)|a^R(t)|$. Symmetrically, for $|a^L(t)| < |a^R(t)|$, we get $(4\rho - 2)|a^L(t)| \geq t_0^R(t)$. In either case, we can derive the following condition on $\rho$.

**FACT 3.2.** The safe tour $T^{safe}(t)$ fulfills inequality (3.1) in Fact 3.1 if and only if $2 \geq \rho \geq \frac{\alpha + \sqrt{\alpha^2 + 4\alpha^2}}{\alpha}.$

We are now ready to describe our algorithm (cf. Algorithm 1). We argued that it is a safe option to follow $T^{safe}(t)$ in order to stay $\rho$-competitive, provided no further requests appear. It will turn out that it is indeed always good enough to follow $T^{safe}(t)$, if possible. However, at time $t$, we may be too far from the extreme that $T^{safe}(t)$ serves first in order to catch up with the safe tour, in which case we have to resort to secondary strategies. If at time $t$ the server cannot reach $T^{safe}(t)$, it instead bases its behavior on the greedy tour as an estimate for $T^{Opt}$. Surprisingly, this estimate turns out to be sufficient to obtain an optimal online algorithm. There are three situations that can occur if the safe tour cannot be reached at time $t$. If the online server is on the same side of the origin as the extreme that $T^{greedy}(t)$ serves first, our algorithm decides to follow the greedy tour. If the online server is on the other side of the origin than the extreme that $T^{greedy}(t)$ serves first, we have to ensure that the condition of Fact 3.1 is not violated. If the tour serving the nearer extreme first satisfies (3.1), the algorithm serves this extreme first, i.e., it serves the extremes in a different order than $T^{greedy}(t)$. Otherwise, we can deduce from (3.1) being violated that we can afford to serve the opposite extreme first, i.e., to follow $T^{greedy}(t)$. 

Theorem 3.1. There is a \((9 + \sqrt{17})/8 \approx 1.64\)-competitive algorithm for closed online TSP on the line.

We obtain the main result of this section by analyzing each of the above cases.

4 Lower Bound for open online TSP

In this section, we consider open online TSP on the line and give a tight lower bound on the best-possible competitive ratio. Note that a lower bound of 2 is obvious: At time 1, we present a request either at \(-1\) or 1, whichever is further away from the online server. The online tour has length at least 2 while the optimum tour has length 1. Remarkably, we are able to show a slightly larger bound that turns out to be tight.

Theorem 4.1. Let \(\rho \approx 2.04\) be the second-largest root (out of the four real roots) of \(9\rho^4 - 18\rho^3 - 78\rho^2 + 210\rho - 107\). There is no \((\rho - \varepsilon)\)-competitive algorithm for open TSP on the line for any \(\varepsilon > 0\).

In the following, we fix any online algorithm \(\text{ALG}\) and \(\rho' \in (2, \rho)\) and describe an adversarial strategy that forces \(|T^{\text{ALG}}|\) to be larger than \(|T^{\text{OPT}}|\) by a factor of at least \(\rho'\). After the first request \(\sigma^R\), which is to the right of the origin, we alternatingly present leftmost and rightmost extreme requests, in the \(i\)-th iteration called \(\sigma^L_i\) and \(\sigma^R_i\), respectively, depending on \(\text{ALG}\)'s behavior. Roughly, a new leftmost request \(\sigma^L_i\) appears whenever the last rightmost request \(\sigma^R_{i-1}\) is served, and a new rightmost request \(\sigma^R_i\) appears when \(\text{ALG}\) has moved close enough to \(\sigma^L_i\). Importantly, we will show that some pair \((\sigma^L_i, \sigma^R_i)\) is critical, in the following sense.

Definition 4.1. We call the last two requests \(\sigma^*_0 = (a^*_0; |a^*_0|)\) and \(\sigma^*_1 = (a^*_1; |a^*_1|)\) of a request sequence with \(\text{sign}(a^*_0) \neq \text{sign}(a^*_1)\) and \(0 < |a^*_0| \leq |a^*_1|\) critical for \(\text{ALG}\) if the following conditions hold:

(i) Both tours \(\text{move}(a^*_0) \oplus \text{move}(a^*_1)\) and \(\text{move}(a^*_1) \oplus \text{move}(a^*_0)\) serve all the requests presented until time \(|a^*_1|\).

(ii) \(\text{ALG}\) serves both \(\sigma^*_0\) and \(\sigma^*_1\) after time \(|a^*_1|\), and \(p_{|a^*_1|}\) lies between \(a^*_0\) and \(a^*_1\).

(iii) Let \(k \in \{0, 1\}\) be such that \(\text{ALG}\) serves \(\sigma^*_k\) before \(\sigma^*_1-k\). Then \(\text{ALG}\) serves \(\sigma^*_k\) no earlier than \(t^* := (2\rho' - 2) \cdot |a^*_1-k| + (\rho' - 2) \cdot |a^*_k|\).

(iv) It holds that \(|a^*_{1-k}|/|a^*_k| \leq 2\).

Indeed, we have the following lemma.

Lemma 4.1. If there is a request sequence with two critical requests for \(\text{ALG}\), we can release additional requests such that \(\text{ALG}\) is not \((\rho - \varepsilon)\)-competitive on the resulting instance.

In the proof, we use the notation from Definition 4.1. We assume that \(\text{sign}(a^*_0) \geq 0\); the other case is symmetric. For the sake of readability, we define \(\sigma^L = (a^L; -a^L) := \sigma^*_{1-k}\) and \(\sigma^R = (a^R; a^R) := \sigma^*_k\).

Conceptually, we want to present additional requests after \(|a^*_0|\) so that \(\text{ALG}\) serves \(\sigma^R\) before \(\sigma^L\). However, it will turn out that serving \(\sigma^R\) first is a
mistake for ALG, compared with using the tour \( T_{\text{LR}} := \) move\((a^L)\) \(\oplus\) move\((a^R)\). Roughly, we make OPT follow the tour \( T_{\text{LR}} \) and then let it continue moving to the right until all requests are served by ALG. Accordingly, we will ensure that all additional requests we introduce coincide with OPT’s position at their release time.

Assume that we could force ALG to serve \( \sigma^L \) immediately after \( \sigma^R \), before serving any additional requests. In this case, we could simply introduce another request at \( a^R \) at time \( |T_{\text{LR}}| \), and, by Definition 4.1 (iii), we would have

\[
|T^{\text{ALG}}| \geq t^* + 2(|a^R| + |a^L|) = (2\rho' - 2) \cdot |a^L| + (\rho' - 2) \cdot |a^R| + 2(|a^R| + |a^L|)
\]

as claimed.

In general, however, ALG may not serve \( \sigma^L \) immediately after \( \sigma^R \), for example by waiting for a while at \( a^R \) – which forces us to postpone the release of additional requests. Of course, ALG needs to start moving towards \( \sigma^R \) at some point to stay competitive if no new requests appear. Our goal is to balance these two effects by introducing one or two new requests.

The additional requests we use depend on the tour that ALG takes after time \( |a^*_1| \). Towards this, let \( t^{**} \) be the earliest possible time that a server starting in \( a^R \) at time \( t^* \) could serve \( \sigma^L \), that is,

\[
t^{**} := t^* + |a^R| + |a^L| = (2\rho' - 2) \cdot |a^L| + (\rho' - 2) \cdot |a^R| + |a^R| + |a^L| = (2\rho' - 1) \cdot |a^L| + (\rho' - 1) \cdot |a^R|.
\]

We characterize the trajectory of ALG at time \( t \geq |a^*_1| \) by the difference between \( t^{**} \) and the earliest possible time that ALG can still serve \( \sigma^L \), if it aborts its tour at time \( t \) and takes the shortest tour serving \( \sigma^R \) (if needed) and then \( \sigma^L \). Formally, for \( t \geq |a^*_1| \), we define

\[
\text{delay}(t) := \begin{cases} 
  t + 2|a^R| + |a^L| - p_t - t^{**}, & \text{if } \sigma^R \text{ not served at } t, \\
  t + |a^L| + p_t - t^{**}, & \text{if } \sigma^R \text{ served at } t \text{ but } \sigma^L \text{ not,}
\end{cases}
\]

It is easy to see that the following properties hold.

**Fact 4.1.** Consider some \( t \) such that \( \text{delay}(t) \) is defined. Let \( \mathcal{T} \) be the set of tours that start in position \( p_t \) at time \( t \) and, if ALG has not served \( \sigma^R \) at time \( t \), that do not visit \( a^L \) before \( a^R \). The following is true:

(i) There is no tour \( T \in \mathcal{T} \) that arrives at \( a^L \) earlier than \( t^{**} + \text{delay}(t) \).

(ii) There is a tour \( T \in \mathcal{T} \) that arrives at \( a^L \) at time \( t^{**} + \text{delay}(t) \).

The following two lemmata state useful properties of the delay function that will be used to define the additional requests.

**Lemma 4.2.** There exists \( W \geq 0 \) with

\[
\text{delay}(\mathcal{T}_{\text{LR}}) + \frac{W}{\rho' - 1} = W.
\]

**Lemma 4.3.** With \( W \) as in Lemma 4.2, ALG serves \( \sigma^R \) no later than time \( |T_{\text{LR}}| + \frac{W}{\rho' - 1} \).

We will show that if we present an additional request at time \( |T_{\text{LR}}| + W/(\rho' - 1) \) (at a distance of \( W/(\rho' - 1) \) to the right of \( \sigma^R \)) and ALG decides to serve \( \sigma^L \) before the new request, the ratio between ALG’s and OPT’s additional costs (Inequality (4.2)) is at least \( \rho' \). If ALG can save time by serving the new request first and does so, we need to present yet another request.

It remains to show that we can define a request sequence (depending on ALG) that ends with a pair of critical requests. We use the following strategy:

- W.l.o.g. \( p_1 \leq 0 \). The first request is \( \sigma^R_0 := (1,1) \).
- Whenever, at some time, called \( t_i^l \) in the following, a request at \( \sigma^R_1 \) gets served, we present the new request \( \sigma^L_i := (a_i^L = -t_i^l; t_i^l) \). Based on \( t_i^l \), we define for \( t \geq t_i^l \) the two functions
  \[
  \ell^L_i(t) := (2\rho' - 3) \cdot t - (3 - \rho') \cdot t_i^l,
  \ell^R_i(t) := (4 - \rho') \cdot t - (2\rho' - 2) \cdot t_i^l,
  \]
  which can as well be viewed as lines in the path-time diagram.
- If at some time \( t_i^R \) after \( t_i^l \), ALG crosses \( \ell^L_i(t) \) or \( \ell^R_i(t) \), we present the request \( \sigma^R_i := (a_i^R = t_i^R; t_i^R) \).
- We stop the procedure when one of the following cases occurs. (The pair \((\sigma_i^L, \sigma_i^R)\) will be shown to be critical in these cases.)

Case 1: ALG serves \( \sigma_i^L \) before \( \sigma_i^R \) if no new requests appear.

Case 2: ALG serves \( \sigma_i^R \) not before time \( (2\rho' - 2) \cdot t_i^R + (\rho' - 2) \cdot t_i^L \) if no new requests appear.

The intuition behind the lines \( \ell_i^L \) and \( \ell_i^R \) is the following: Suppose the position of ALG at \( t_i^L \) is on or to the right of \( \ell_i^L \) and ALG decides to serve \( \sigma_i^L \)
before \( \sigma_i^R \) in case no new requests appear after \( t_i^R \). Then the pair \((\sigma_i^L, \sigma_i^R)\) satisfies Definition 4.1 (iii). The symmetric statement holds for \( \ell_i^R \). The following lemma ensures that, in each iteration, we obtain a (non-necessarily critical) pair of unserved requests \((\sigma_i^L, \sigma_i^R)\) and that Definition 4.1 (iv) is fulfilled.

**Lemma 4.4.** Let \( i \geq 1 \). At time \( t_i^L \), ALG is to the right of \( \ell_i^L \) and \( t_i^L \) and crosses one of them after \( t_i^L \) and before it serves \( \sigma_i^L \). We also have that \( t_i^L \leq 2t_i^L \).

In the proof of Theorem 4.1, we show that Case 1 or 2 eventually occurs, and we formalize the above intuition to show along with Lemma 4.4 that the requests \((\sigma_i^L, \sigma_i^R)\) are indeed critical.

**5 Algorithm for open online TSP**

In Algorithm 2, we describe an algorithm for the open online TSP on the line which achieves a competitive ratio of \( \rho \approx 2.04 \), matching the lower bound presented in Section 4.

**Theorem 5.1.** Algorithm 2 is \( \rho \)-competitive with \( \rho \approx 2.04 \) being the second-largest root of the polynomial \( 9\rho^4 - 18\rho^3 - 78\rho^2 + 210\rho - 107 \).

In the following, we discuss the different cases that can occur in Algorithm 2 and give an intuition of their interplay. Algorithm 2 is called every time a new request is released. It then computes a new tour serving the current extreme requests and thus also all other requests between these extremes. It is important that we wait in certain cases to protect against the release of new extremes that force us to serve the extremes in a different order than we initially chose. Algorithm 2 lets the server return to the origin and wait there whenever possible, and moves to the extremes as late as possible. Intuitively, staying as close to the origin as possible has the benefit that the algorithm can delay the choice in which order to serve the extremes for as long as possible. We observe in the lower bound construction (Section 4), that a \( \rho \)-competitive algorithm may not serve a request too early, i.e., \( |p_i|/t \) is bounded when a request is served. The exact bound on this ratio is computed in the proof of Lemma 4.4 and coincides with the bound in Lemma 5.1 for Algorithm 2.

**Lemma 5.1.** The position of the server in a tour computed by Algorithm 2 satisfies

\[
|p_i| \leq \frac{3\rho - 5}{-3\rho^2 + 9\rho - 4} \approx 0.58
\]

for all times \( t \geq 0 \).

We decide the order in which we serve the extremes and possibly wait based on the current position \( p_i \) of the server, the current time \( t \) and the release times and positions of the single or the two extremes. The server always tries to move back to the origin and wait there as long as possible, i.e., it follows the tour \( T_0 := \text{move}(0) \oplus \text{wait until}(\infty) \) for as long as possible. We use the notation “\( T_0 \), until\( (\text{condition}(\tau)) \)” to denote the tour that follows \( T_0 \) until the first time \( t_0 \) that satisfies “\( \text{condition}(t_0) \)” . Note that this may happen before the server reaches the origin.

We now discuss the different cases of the algorithm step-by-step. Let us first consider the simplest case where only one extreme request \( \sigma_1 = (a_1; t_1) \) is present (cases (P1) and (E1) in Algorithm 2). The offline optimum \( \text{OPT} \) obviously cannot finish before \( t_1 \) in this case. In order to guarantee \( \rho \)-competitiveness it is therefore sufficient to serve \( \sigma_1 \) at time \( \rho t_1 \). Hence, we can afford to move to the origin and wait until the equation \( t + |p_1 - a_1| = \rho t_1 \) is satisfied for the current time \( t \) and position \( p_1 \). This ensures that \( |p_i|/t \) is bounded as stated in Lemma 5.1 (this is not clear and formally proved in the full version of the paper) and that we have the option to change our tour without much additional cost if an extreme on the other side is released later on. We call the resulting tour in this case, the preferred tour (case (P1)). It can happen, however, that we are only able to serve \( \sigma_1 \) later than time \( \rho t_1 \), i.e. \( t + |p_1 - a_1| > \rho t_1 \). This means that the until-condition is already satisfied and the server serves \( \sigma_1 \) immediately. We call this tour the enforced tour (case (E1)). The makespan of Algorithm 2 in this case is \(|\text{Algorithm}| = t_1 + |p_1 - a_1| \). But we have \(|\text{Algorithm}| \geq t_1 \) and also \(|\text{Algorithm}| \geq |p_1 - a_1| \) as Algorithm 2 only visits points on the real line that must also be visited by \( \text{OPT} \) at some time. Overall, we have \(|\text{Algorithm}| \leq 2 \cdot \text{OPT} \), implying \( \rho \)-competitiveness.

Now, consider the case where two extremes \( \sigma_1 = (a_1; t_1) \) and \( \sigma_2 = (a_2; t_2) \) are present. The two extremes define an interval

\[
[a^L(t), a^R(t)] = [\min\{a_1, a_2\}, \max\{a_1, a_2\}].
\]

Note that \( a^L(t) < p_i < a^R(t) \) holds by definition of extreme requests.

If \( 0 \notin [a^L(t), a^R(t)] \) (case (O) in Algorithm 2), we let \( \sigma_1 \) denote the request closer to the origin, i.e., \( |a_1| \leq |a_2| \). We then immediately serve \( \sigma_1 \) in order to ensure that Lemma 5.1 holds and \( |p_i|/t \) stays small. We know that the offline optimum \( \text{OPT} \) cannot finish before time \( t_2 \). After serving \( \sigma_1 \) it is therefore safe to return to the origin and wait as long as we can to reach \( a_2 \) at time \( \rho t_2 \). We can thus follow the tour \( T_0 \) after serving \( \sigma_1 \) at time \( t + |p_1 - a_2| \geq \rho t_2 \). Possibly, this equation is already satisfied from the start and
we serve $\sigma_2$ immediately.

Next, consider the case that $0 \in [a^L(t), a^R(t)]$ (cases $(P2)$, $(A2)$ and $(E2)$). Let now $\sigma_1$ be the request released first, i.e. $t_1 \leq t_2$, and $t = t_2$ be the current time. We have a lower bound of $|T^{opt}| \geq t_1 + |a_1| + |a_2|$, irrespective of whether $Opt$ serves $\sigma_1$ or $\sigma_2$ first. If we serve $\sigma_1$ first, which we call the preferred tour (case $(P2)$), we ensure that the tour produced by Algorithm 2 is not longer than $\rho |T^{opt}|$ by satisfying the inequality

$$t + |p_t - a_1| + |a_1| + |a_2| \leq \rho(t_1 + |a_1| + |a_2|). \quad (5.4)$$

Now assume that Opt serves $\sigma_2$ first and a new request $\sigma'_1 = (a_1; t_2 + |a_1| + |a_2|)$ appears at the same position as $\sigma_1$ when Opt arrives there. The new request does not increase the cost for $Opt$, which is still only lower bounded by $|T^{opt}| \geq t_2 + |a_1| + |a_2|$. But our algorithm, which served $\sigma_1$ first, may be closer to $\sigma_2$ at the time when this new request appears and now has to go all the way back to the position of $\sigma_1$ after serving $\sigma_2$. The intuition why it is sufficient to protect against this worst-case is that if $\sigma'_1$ appears at a position far away from the origin, then this additional distance has to be traveled by $Opt$ as well, and if $\sigma'_1$ appears closer to the origin, it only benefits our algorithm. In order to ensure $\rho$-competitiveness in this scenario, the following inequality has to also be satisfied:

$$t + |p_t - a_1| + 2(|a_1| + |a_2|) \leq \rho(t_2 + |a_1| + |a_2|). \quad (5.6)$$

If we now define

$$L^{\sigma_1, \sigma_2} := \min\{\rho t_1 + \rho - |a_1| + (\rho - 1)|a_2|; \rho t_2 + (\rho - 2)|a_1| + (\rho - 2)|a_2|\},$$

then Inequalities (5.4) and (5.5) are simultaneously satisfied if and only if $t + |p_t - a_1| \leq L^{\sigma_1, \sigma_2}$. If this latter inequality is satisfied with some slack, we again follow the tour $T_0$ until it becomes tight, so that $|p_t|/t$ stays small and we are more flexible to change our tour when new requests appear.

In case the conditions for the preferred tour are not satisfied, we try to serve $\sigma_2$ first. This is called the anticipated tour (case $(A2)$). The inequalities that need to be satisfied in this case are the same as those for the preferred tour with $\sigma_1$ and $\sigma_2$ exchanged. Moreover, to ensure that $|p_t|/t$ is bounded as claimed in Lemma 5.1, the following inequality also needs to be satisfied if Algorithm 2 serves $\sigma_2$ first:

$$|a_2| \leq \frac{3\rho - 5}{(2\rho - 2)(7 - 3\rho)} (\rho t_1 + (\rho - 2)|a_1|) \approx 1.21 \cdot t_1 + 0.02 \cdot |a_1|. \quad (5.6)$$

Intuitively, the inequality ensures that $\sigma_2$ is not too far from the origin compared to $\sigma_1$, as otherwise we
would need to start too early to serve the extreme \( \sigma_2 \)
and would violate the bound on \( |p_i|/t \).

Lastly, in case neither the conditions for the preferred tour nor the anticipated tour are met, we
immediately serve the earlier request \( \sigma_1 \) first and
\( \sigma_2 \) directly afterwards. This is called the enforced
tour (case (E2)). The main challenge in showing \( \rho \)-
competitiveness in the analysis of the algorithm is to
derive a better lower bound on \( \text{Opt} \) in this case by
considering extremes that must have been released before.

6 Online Dial-A-Ride on the line

In this section we give a \((1 + \sqrt 2)\)-competitive algorithm
for (preemptive) open online Dial-A-Ride on the line. Let \( T_{\text{Opt}}^S \)
denote the optimal tour over a
set of requests \( S \), starting at position \( p_0 = 0 \), and
let \( R_t \) denote the set of released but not yet deliv-
ered requests at a time \( t \). Our algorithm works as
follows (cf. Algorithm 3): The server stays at position
\( p_0 \) until the first request arrives. With every new
arriving request, the server stops its current tour and
returns to \( p_0 \), or stays at \( p_0 \) if it is already at the
origin. The server starts following the tour \( T_{\text{Opt}}^{R_t} \)
at time \( \sqrt 2 \cdot |T_{\text{Opt}}^{R_t}| \). By “unload” we denote the opera-
tion of unloading all requests the server is currently
carrying at the current position. Formally, each such
request \((a, b; t')\) is changed to \((p_i, b; t')\).

**Algorithm 3:** For the open online Dial-A-Ride problem with \( c \geq 0 \) fixed.

**Input:** new request \( \sigma \), current position \( p_i \),
unserved requests \( R_t \).

**Output:** an open tour starting at \( p_i \) and
serving all requests in \( R_t \).

**Return:** unload \( \oplus \) move \((p_0)\)
\( \oplus \) waituntil \( \left( \sqrt 2 \cdot |T_{\text{Opt}}^{R_t}| \right) \).

We show that we get back to the origin in time
whenever a new request is released.

**Theorem 6.1.** Algorithm 3 is \((1 + \sqrt 2) \approx 2.41\)
competitive for the preemptive open online Dial-A-
Ride problem with capacity \( c \geq 1 \).

**Note 6.1.** Algorithm 3 can easily be modified to
solve open Dial-A-Ride in general metric spaces
and upholds the same competitive ratio. The proof
remains the same.

We also provide a lower bound for non-preemptive
closed Dial-A-Ride on the line that improves the
lower bound of 1.70 from [2].

**Theorem 6.2.** No algorithm for the non-preemptive
closed Dial-A-Ride problem on the line with fixed
capacity \( c \geq 1 \) has competitive ratio lower than \( \rho =
1.75 \).

7 The offline problem

Psarafitis et al. [21] show that open offline TSP on the
line with release dates can be solved in quadratic time.
For the closed variant they claim that the optimal
tour has the structure [21, pp. 215–216]:

\[
\text{waituntil()} \oplus \text{move}(A^R) \oplus \text{move}(A^+) \oplus \text{move}(p_0)
\]

or

\[
\text{waituntil()} \oplus \text{move}(A^+) \oplus \text{move}(A^R) \oplus \text{move}(p_0).
\]

Here the waiting time at the origin is chosen maximally
such that all requests are still served. We contradict
this claim by showing that an optimal server tour may
need to turn around arbitrarily many times.

**Theorem 7.1.** For every \( k \in \mathbb{N} \), there is an
instance of closed TSP on the line such that any optimal
solution turns around at least \( 2k \) times.

Next, we show a dynamic program that solves
closed offline TSP on the line in quadratic time. It
is inspired by the one of Psarafitis et al. [21] for the
open variant and a dynamic program of Tsitsiklis [22]
for the same problem with deadlines instead of release
times.

The algorithm (cf. Algorithm 4) relies on the fact
that an optimal server tour has a ‘zig-zag shape’ with
decreasing amplitude.

We index the requests by increasing positions. Then we compute for each index pair \( i < j \) the
completion time of two tours serving all requests to
positions smaller than position \( a_i \) and all requests
to positions larger than position \( a_j \). We use \( C^+_{i,j} \)
for the best such tour ending at position \( a_j \) and \( C^-_{i,j} \)
for the best one ending at \( a_i \). Starting with the
largest difference \( j - i \) and iteratively decreasing it,
we recursively compute \( C^+_{i,j} \) and \( C^-_{i,j} \).

**Theorem 7.2.** Algorithm 4 computes the minimum
completion time of a server tour for offline TSP on
the line in time \( O(n^2) \).

For the non-preemptive Dial-A-Ride problem on the
line we show that the open and closed variant with
release times are NP-hard. Without release times,
we prove they are NP-hard for capacity \( c \geq 2 \). Our
reductions are from the circular arc coloring problem,
which is also used in a reduction for minimizing the
sum of completion times of Dial-A-Ride on the line
with capacity \( c = 1 \) [9].
Algorithm 4: Dynamic Program for Closed Offline TSP on the line.

Input: A set of requests $\sigma_i = (a_i; t_i)$ with $t_i \geq |a_i|$ for $-\ell \leq i \leq r$ and $a_i < a_{i+1}$ for $-\ell \leq i < r$.
Output: The minimum completion time $C_{\text{max}}$ of a tour.

$C_{-\ell,r}^+ \leftarrow t_r$,
$C_{-\ell,r+1}^- \leftarrow t_{-\ell}$.

for $i = -\ell, ..., r$ do
  $C_{i,r+1}^+ \leftarrow \max\{t_i, C_{i-1,r+1}^- + a_i - a_{i-1}\}$.
end

for $j = r, ..., \ell$ do
  $C_{-\ell-1,j}^+ \leftarrow \max\{t_j, C_{-\ell-1,j+1}^- + a_{j+1} - a_j\}$.
end

for $d = r + \ell, ..., 0$ do
  for $i = -\ell, ..., r - d$ do
    $j \leftarrow i + d$.
    $C_{i,j}^- \leftarrow \max\{t_j, \min\{C_{i-1,j}^+ + a_j - a_i, C_{i-1,j}^- + a_i - a_{i-1}\}\}$.
    $C_{i,j}^+ \leftarrow \max\{t_j, \min\{C_{i,j+1}^+ + a_{j+1} - a_j, C_{i,j+1}^- + a_j - a_i\}\}$.
  end
end

return $C_{\text{max}} = C_{0,0}^+$.

Theorem 7.3. The non-preemptive open and closed offline Dial-A-Ride problem on the line are NP-complete. For capacity $c \geq 2$ this even holds when all release times are 0.

In the classification of closed dial-a-ride problems in [9], offline TSP on the line is the problem $1|s=t, d_j|\text{line}|C_{\text{max}}$. De Paepe et al. [9] claim Tsitsiklis [22] shows a polynomial algorithm, but this is for the open version. We solve the closed variant by giving a polynomial algorithm (Theorem 7.2) as well as a counterexample to the algorithm of Psaraftis et al. [21]. Our Theorem 7.3 shows the problem $1, \text{cap}\{d_j\}|\text{line}|C_{\text{max}}$ in the same classification scheme is NP-hard. While this is implicitly claimed in [9], no proof is given. For the generalization to arbitrary capacity but without release dates, $1||C_{\text{max}}$, Guan [13] showed hardness for capacity $c = 2$ and our new hardness proof handles any capacity $c \geq 2$.

References


