Earliest Deadline First is a 2-approximation for DARP with Time Windows

Barbara M. Anthony[1][0000−0002−2493−1251], Christine Chung[2][0000−0003−3580−9275], Ananya Das[3][0000−0001−9445−1475], and David Yuen[4][0000−0001−9827−0962]

[1] Southwestern University, Georgetown TX 78626, USA anthonyb@southwestern.edu
[2] Connecticut College, New London CT 06320, USA cchung@conncoll.edu
[3] Middlebury College, Middlebury VT 05753, USA adas@middlebury.edu
[4] Kapolei HI 96707, USA yuen888@hawaii.edu

Abstract. Dial-a-Ride problems (DARP) require determining a schedule to efficiently serve transportation requests in various scenarios. We consider a variant of offline DARP in a uniform metric space where requests have release times and deadlines, and are all of equal duration and value. The goal is for a single unit-speed, unit-capacity server to serve as many requests as possible by an overall time limit, and this problem is NP-hard. We show that a natural greedy algorithm, Earliest Deadline First, is a 2-approximation, and this is tight.

Keywords: Approximation algorithms · Deadline scheduling · Dial-a-Ride problems.

1   Introduction

The widely studied Dial-a-Ride Problem (DARP) requires scheduling one or more servers to complete a sequence of rides, each consisting of a pickup location (or *source*) and delivery location (or *destination*). Common optimality criteria include minimizing makespan (i.e., the time the server has completed the last request), minimizing the average flow time (i.e., the difference in a request's completion and release times), or maximizing the number of served requests within a specified time limit. In many variants *preemption* is not allowed, so if the server begins to serve a request, it must do so until completion. Applications include the transport of people and goods, including package delivery services, ambulances, ride-hailing services, and paratransit services. For an overview of DARP and its many variants, please refer to the surveys [12, 22, 17].

In this work we study offline DARP on the uniform metric space with a single unit-capacity server, where each request has a source, destination, release time, and deadline. Requests can be served only between their release time and their deadline and may not be preempted. The server also has a specified time limit T after which no more requests can be served, and the goal is to maximize the number of requests served within T. This variant may be useful for settings

where several equal-length requests must be completed within a deadline and each one has a specified time window for completion.

For the remainder of this paper, we refer to this request-maximizing, **T**ime-bounded **DARP** variant, where each request has a specified **T**ime **W**indow (its release time to its deadline) as TDARPTW. TDARPTW is NP-hard because a special case of the problem without time windows, TDARP, where each request can be considered to have release time 0 and deadline T, has already been shown to be NP-hard in [1]. At any time t, given the current server location, we refer to a request as *servable* if it has not already been served, and can feasibly be served by its deadline. In this work, we show that the algorithm, which we refer to as EARLIEST DEADLINE FIRST (EDF), that continuously serves the servable request with the earliest deadline, has approximation ratio at most 2 for TDARPTW.

A common objective studied in previous DARP work is to minimize the time needed to serve all requests, as inspired by the original objective of the Traveling Salesperson Problem, since DARP is a generalization of TSP [16, 7]. We note that this more classical DARP objective can be reduced to the time-bounded variant we study here, which has the objective of maximizing the number of requests served by time T. A solution for our time-bounded request-maximizing TDARP can be invoked a polynomial number of times to find the minimum time-bound where all requests can be served. Hence TDARP is at least as hard as its corresponding more classical DARP variants.

There has been limited prior work on approximation algorithms for TDARP in the offline setting. The work in [4] gives a $O(\log n)$-approximation for the closely related Vehicle Routing with Time Windows problem where n nodes in a metric space have release times and deadlines and the goal is to visit as many nodes as possible within their time windows. The work of [1] presented a 3/2-approximation algorithm for the uniform metric version of the problem without the constraint of time windows, or equivalently, where the time window for each request was assumed to be $[0, T]$. The work of [2] showed that the Segmented Best Path algorithm of [10, 11] is a 4-approximation for TDARP on non-uniform metric spaces. That same work also shows that a greedy algorithm that repeatedly serves the fastest set of k remaining requests has an approximation ratio of $2 + \lceil \lambda \rceil / k$, where λ denotes the aspect ratio of the metric space.

In much of the voluminous and wide-ranging DARP literature, which is most often generated by applied researchers in operations and management, the term "time windows" often refers to slightly different notions than in our model, where we simply have a release time and a deadline per request. DARP in settings with time windows has been extensively studied with various parameters [18, 23, 20, 15], and the particular definition of time windows in our model has also been studied in works such as [26, 27, 25], which unlike all the empirical research in DARP, lies instead in the domain of competitive analysis of online DARP problems. In this domain, there have been a number of notable recent developments, albeit with different objectives [3, 5, 6].

ODAPRTW, investigated by [26], is the same as our TDARPTW problem, except the server has no overarching time limit, and their problem is online,

so the requests are not known a priori, but instead arrive over time. They also assume the time windows are of uniform length for all requests, and their goal is to maximize the number of requests that meet their deadlines. They give an algorithm with competitive ratio $(2 - \Delta)/(2\Delta)$, where Δ denotes the diameter of the metric space. In [25] they consider time windows of non-uniform length, as we do, but in the online setting. They find that a greedy-by-deadline algorithm is 3-competitive in the uniform metric.

DARP problems also generalize scheduling problems, as they require requests to be served with the additional constraint of a metric space in which the server must move to reach each request it wishes to serve. Scheduling requests with a given deadline for each request is part of the setting in many classical problems studied in scheduling theory literature, and the *Deadline Scheduling Problem*, for example, is well-known to be (strongly) NP-hard [19].

The EDF scheduling algorithm is a simple greedy approach that is well-known to be effective in many different contexts—offline, online, and in real-time systems—for various scheduling objectives where jobs to be scheduled have deadlines (for a sample of such settings, see [19, 8, 13, 14, 9, 24]). Due to space constraints, we omit a more detailed discussion of the scheduling literature here. To our knowledge, our work is the first to present an approximation guarantee for greedy EDF scheduling of DARP requests with release times and deadlines.

2   Formalizing the Problem and Algorithm

The input for an instance of our problem, TDARPTW, consists of a uniform metric space, an origin, a set of requests S, and an overall time limit T. The origin o, a point in the metric space, indicates where the server is at time 0; a request may start at o but need not. Each request is an ordered tuple (s, d, a, b) consisting of the source s, destination d, release time a, and deadline b. We focus on the offline variant, where all information is known at time 0.

We assume without loss of generality that all time values (e.g., release times, deadlines, T) are integers. We do not allow preemption so if the server begins serving a request, it will serve it to completion. For any request i, let a_i and b_i denote the release time and deadline of i, respectively, with $0 \le a_i < b_i$. We refer to $[a_i, b_i]$ as the *time window* for request i. The goal for TDARPTW is to maximize the number of requests that a single unit-capacity server can serve within their time windows by the time limit T.

For an algorithm ALG and a TDARPTW instance I, ALG(I) denotes the schedule created by ALG on I, i.e., the action prescribed by ALG for the server at each time unit, and OPT(I) denotes an optimal schedule on I. $|$ALG$(I)|$ and $|$OPT$(I)|$ denote the number requests served by ALG and OPT, respectively, on instance I. When the instance is clear from the context, (I) may be dropped.

We define a request to be "servable" at a given time if it can be reached and served within its time window. We also define "drive" below, similarly to [1]. In the uniform metric space, all drives (empty or service) each take one time unit. We emphasize that because the schedule is part of the definition of a drive,

serving the same request in EDF(I) and OPT(I), even during the same time unit, is considered two distinct drives.

Definition 1. *For a request i and a time t, let $h_i(t)$ be an indicator variable where $h_i(t) = 0$ if the server is at the source of request i at time t and $h_i(t) = 1$ otherwise. We characterize request i as **servable** at time t if request i has not already been served by time t and $a_i \leq t + h_i(t)$ and $b_i \geq t + h_i(t) + 1$.*

Definition 2. *A **drive** refers to one movement of the server from one point in the metric space to a second (not necessarily distinct) point at a specific time in a given schedule (e.g. EDF or OPT). A drive that serves a request is a **service drive**. A drive that does not serve a request but simply re-positions the server or allows it to remain at the current location for one time unit is an **empty drive**.*

2.1   The Earliest Deadline First (EDF) Algorithm

One might be inclined to think it is trivial to find an algorithm that serves a request every other time unit, giving a 2-approximation. However, due to the constraints of the release times and deadlines, such schedules may not always exist or be straightforward to find.

The focus of this paper is on an algorithm inspired by the well-established earliest deadline first greedy algorithms from the scheduling literature (see, e.g., [19, 24]). Informally, we greedily consider all servable requests, serving one with the soonest deadline, breaking ties arbitrarily, as long as time remains. Our EDF algorithm is formally described in Algorithm 1. Note that EDF does not simply sort by earliest deadline, as a request is servable only if it has been released. Furthermore, EDF will first consider requests that can be served in the next two time units rather than only those in the next time unit, as only those requests that begin where the server is currently located can be served in the next time unit. Requiring the server to prioritize requests at its current location could result in serving requests with later deadlines while requests with earlier deadlines are available. This is not necessarily beneficial as we show in Theorem 4.

3   Our Results

We first lower bound the performance of EDF by showing that there exist instances of TDARPTW such that $|\text{EDF}| = |\text{OPT}|/2$.

Theorem 1. *There exist arbitrarily large instances of TDARPTW such that $|\text{OPT}| = 2 \cdot |\text{EDF}|$.*

Proof. Please see Figure 1. Consider an instance which consists of T requests, each with release time 0 and deadline T, where the destination of one request is the source of the next request, and $T/2$ requests, each with release time 1 and deadline $T - 1$, where all the sources and destinations are distinct points in the metric space. No requests have their source at the origin.

Algorithm 1 The Earliest Deadline First (EDF) algorithm

1: Input: Metric space of points, set S of requests, time limit T, origin o.
2: **while** $t \le T - 2$ **do**
3:    **if** there are no servable requests (recall definition of servable) **then**
4:        $t = t + 1$
5:    **else**
6:        Choose a request, r, with the earliest deadline among all servable requests.
7:        Let s and d denote the source and destination, respectively, of r.
8:        **if** the server is at s **then**
9:            Serve r during time $[t, t + 1]$. Set $t = t + 1$.
10:       **else**
11:           During $[t, t + 1]$, move the server to s and during $[t + 1, t + 2]$ serve r.
12:           Set $t = t + 2$.
13:       **end if**
14:   **end if**
15: **end while**
16: **if** $t = T - 1$ and there is a servable request **then**
17:    Serve any such servable request during $[t, T]$.
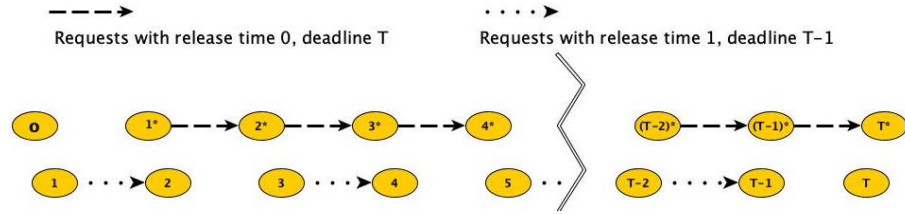18: **end if**

Fig. 1. An instance in which for T odd, $|\text{OPT}| = 2 \cdot |\text{EDF}|$. The server starts at the origin o. All requests are represented by arrows, either dashed or dotted. The top row consists of T requests with dashed arrows, each with release time 0 and deadline T, all of which can be served by OPT. The bottom row illustrates $T/2$ requests with dotted arrows, each with release time 1 and deadline $T - 1$, which will be served by EDF.

OPT can serve all $T - 1$ of the requests with release time 0 and deadline T; no better solution is possible in time T as there is no request beginning at the origin. Thus, $|\text{OPT}| = T - 1$. EDF chooses a request with deadline $T - 1$ over any request with deadline T. Accordingly, EDF serves all $(T-1)/2$ requests with deadline $T-1$ and must do an empty drive between every pair of requests, giving $|\text{EDF}| = (T - 1)/2 = |\text{OPT}|/2$. $\square$

For the upper bound, we will show by induction that $|\text{OPT}(I)| \le 2|\text{EDF}(I)|+1$ for any input instance I. The most involved case of the proof is when $\text{EDF}(I)$ has as least two consecutive empty drives, so we define terminology for that scenario and a given instance I, enabling us to derive various facts that we can then use in the overall proof. Key to our work is considering what requests are or could have been servable before the first time EDF makes two consecutive empty drives,

since without two consecutive empty drives, EDF would be serving at least one request every other time unit, immediately giving us the ratio of 2. Accordingly, let the earliest such occurrence of two consecutive empty drives by EDF(I) be during the interval $[\tau, \tau + 2]$, for some time $\tau \leq T - 2$. Let w be the number of requests EDF(I) has served by time τ.

We now focus on requests that are served before τ, and in particular, requests whose deadlines are after τ, since our induction will be based on a smaller instance created by removing such requests. Let E^* be the set of empty drives that OPT(I) makes during $[0, \tau]$. Let R^* be the set of requests with deadline $\geq \tau + 2$ that OPT(I) serves during $[0, \tau + 2]$. We define a set R to be analogous to R^* except served by EDF(I) instead of by OPT(I). Formally, let R be the set of requests with deadline $\geq \tau + 2$ that EDF serves during $[0, \tau + 2]$. Note that the deadline requirement for R^* requests ensures that $R^* \subseteq R$, since if a request in R^* was not served by EDF during $[0, \tau]$, it could have then been served during $[\tau, \tau + 2]$ in which EDF makes two empty drives. The size of the set $R - R^*$ is important in the inductive proof of Theorem 2.

We further partition the requests in R into two sets depending upon whether or not they were served in EDF(I) after an empty drive. Formally, define disjoint sets X and Y with $X \cup Y = R$ as follows: $Y \subseteq R$ are requests served in EDF(I) after time interval $[0, 1]$ that do not immediately follow another request in EDF(I); they must follow an empty drive in EDF(I). $X \subseteq R$ are requests which are either served at time 0 or immediately follow another request in EDF(I), which could have been in X but could likewise have been in Y or need not have been in R at all if its deadline was less than $\tau + 2$.

Let the set Q be all requests served in EDF(I) by time τ that were not preceded in EDF(I) by an empty drive. Thus, all requests in Q are either served at time 0 or immediately follow another request served by EDF. Q is a superset of X, as unlike X, the requests in Q are not constrained to have a deadline $\geq \tau + 2$.

Since τ is the first time at which two consecutive empty drives occur in EDF's schedule, every empty drive by EDF in $[0, \tau]$ is followed by a request served. We can thus add the number of requests served by EDF by time τ, or w, to the number of empty drives by EDF prior to τ, or $w - |Q|$, to get

$$\tau = 2w - |Q| \tag{1}$$

Understanding how EDF differs from OPT allows us to bound EDF's performance. As such, we construct an alternating sequence of drives that we call a *trace* that allows us to catalog EDF's non-optimal choices, as well as what OPT was doing when EDF served a different request, "tracing" through the two parallel schedules to the roots of the discrepancies between the two schedules. See Figure 2 for an example.

Definition 3. *An **alternating trace**, or simply a **trace**, is a sequence of distinct drives that alternates between drives of OPT and drives of EDF. A trace is constructed using the rules below, beginning with the initialization in Rule 1 and then iterating between Rule 2 and Rule 3 as applicable until a termination*

*condition (Rule 4-Rule 6) is reached. A **maximal alternating trace** is a trace that is not a proper subsequence of any other trace.*
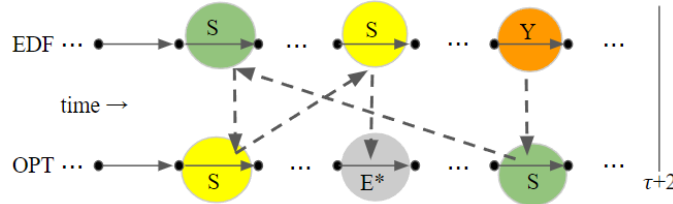

Fig. 2. An alternating trace. This figure depicts the schedule of drives in EDF in parallel with that of OPT, with time increasing to the right. Each drive (empty or service) is represented by a directed edge to the right between two points. The trace itself is represented by the dashed edges leading from a drive in one schedule to a drive in the other. The trace shown here starts with a drive in EDF serving a request from $Y \subseteq R$ shown in orange, and ends with an empty drive of OPT (in E^*) shown in gray. Drives that are shaded the same color serve the same request.

Rule 1 Begin the trace with a drive in EDF from R or a drive in OPT from R^*.

Rule 2 If the drive most recently added to the trace is a request served by OPT, it is immediately followed in the trace by the EDF drive serving the same request. (Note that we know EDF has indeed served the same request at some point due to Lemma 1 below.)

Rule 3 If the drive most recently added to the trace is a drive EDF did at some time t and is preceded in EDF by an empty drive, then the next drive in the trace is the drive OPT did at time t.

Rule 4 If the drive most recently added to the trace is one that EDF did at time t that was not preceded in EDF by an empty drive at time $t - 1$, then the trace terminates after said drive.

Rule 5 An empty drive in OPT ends the trace.

Rule 6 If the preceding rules ever result in an attempt to add a drive that is already part of the trace, the trace instead terminates (e.g., if there is a request served by both EDF and OPT at the same time, and applying Rule 2 would cause the same EDF drive to be added to the trace a second time). This rule prevents the same drive within a schedule from being repeated in the trace.

Maximal alternating traces are disjoint from one another due to the deterministic nature of their construction. We use the following lemma for Rule 2, above, as well as for our main result in Theorem 2.

Lemma 1. *In a maximal trace, every request served by a drive of OPT must also be served by a drive of EDF, and these drives never occur past $\tau + 2$.*

Proof Idea. Suppose there is a trace with a drive of OPT before time $\tau + 2$ serving a request $q \in R^*$. Then EDF could have served q at time $\tau + 1$, but did not because EDF has two empty moves during $[\tau, \tau + 2]$. Therefore EDF must have already served q before time τ, implying $q \in R$. The difficult case is when $q \notin R^*$, having a deadline before $\tau + 2$, and hence also $q \notin R$. For this case, we strengthen the inductive hypothesis, and show any request satisfying certain time window constraints (which any request served by OPT must also satisfy) will be served by EDF. Please find the proof in the Appendix. □

To aid in categorizing maximal traces, in Lemma 2 we state their possible termination types. Note that every request in R and R^* appears exactly once in some maximal trace. To relate $|EDF|$ to $|OPT|$ we endeavor to find a relation between the sizes of $R - R^*$, Q, and E^*. An increase in $|R - R^*|$ can decrease what EDF serves after time τ, an increase in $|Q|$ increases what EDF serves before time τ, and an increase in $|E^*|$ decreases what OPT serves before time τ.

Lemma 2. *Other than traces that consist of a drive from OPT and a drive from EDF that serve the same request in the same time unit[†], any maximal alternating trace ends in Q or E^*.*

Proof. By Lemma 1 and Rule 2, a maximal trace cannot end at a request served by OPT except in the case of Rule 6. Thus the only way of ending a trace is if EDF serves a request in Q (Rule 4), or if OPT does an empty drive (Rule 5). □

Let C denote the collection of all maximal alternating traces. Due to Lemma 2, we may partition C into six categories C_0 through C_5 as described below. (Also see Figure 3.) For the remainder of this work, the term "trace" always refers to maximal traces.
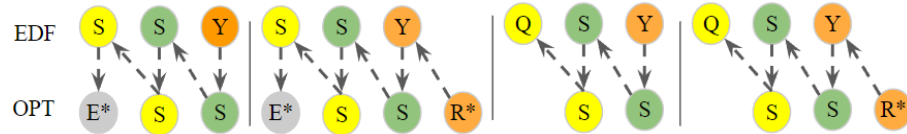

Fig. 3. Small examples depicting four general categories of traces (from left to right): C_1, C_2, C_3, and C_4. Recall that $Y \subseteq R$. To avoid crossing lines, these are samples of traces that proceed from later in the schedules to earlier, but as shown in Figure 2, the upward edges may, more generally, also point forward in time rather than backward. (Also see Table 1 in the Appendix.)

Trace categories C_0 and C_5 are not shown in Figure 3, as they are special cases where the trace includes only a single request.
C_0: traces consisting solely of requests from X (terminated immediately due to Rule 4)

[†]In this special case, represented by category C_5, the trace ends in R^* or Y.

C_1: traces beginning with a request from $Y \subseteq R$ and ending with an empty drive from E^* (terminated due to Rule 5); one such trace was also seen in Fig. 2.

C_2: traces beginning with a request from R^* and ending with an empty drive from E^* (terminated due to Rule 5)

C_3: traces beginning with a request from $Y \subseteq R$ and ending with a request from Q (terminated due to Rule 4)

C_4: traces beginning with a request from R^* and ending with a request from Q (terminated due to Rule 4)

C_5: traces consisting only of two drives: the same request served at the same time in both R^* and Y (terminated due to Rule 6)

Definition 4. *For any set of requests P, we define P_i to be the number of requests from P that occur in a trace from category C_i.*

We derive the following facts about the size of R^* by considering trace categories C_0 to C_5. From Lemma 1, we know that $R_i^* \le R_i = X_i + Y_i$ for any trace category $i = 1 \ldots 5$, and we also know from Rule 2 that every request of X_i or Y_i was preceded in the trace by a request of R_i^*, so $R_i^* = X_i + Y_i$, unless the trace started with an additional drive from EDF. So for trace categories that start with an initial EDF request (from Y, in the case of C_1 or C_3, or from X, in the case of C_0) we must subtract $|C_i|$ in computing the size of R^*.

Fact 0: $R_0^* = X_0 + Y_0 - |C_0|$. (Here, $Y_0 = 0$ and $X_0 = |C_0|$, so $R_0^* = 0$.)
Fact 1: $R_1^* = X_1 + Y_1 - |C_1|$. (Here, $X_1 = 0$.)
Fact 2: $R_2^* = X_2 + Y_2$. (Here, $X_2 = 0$.)
Fact 3: $R_3^* = X_3 + Y_3 - |C_3|$.
Fact 4: $R_4^* = X_4 + Y_4$.
Fact 5: $R_5^* = X_5 + Y_5$. (Here, $X_5 = 0$.)

We are now ready to prove the main theorem of this work.

Theorem 2. $|\text{OPT}(I)| \le 2|\text{EDF}(I)| + 1$ *for any input instance I of TDARPTW.*

Proof. We proceed by induction on $|\text{EDF}(I)| + T$. Base step: $|\text{EDF}(I)| + T = 1$. In this case $T = 1$ and $|\text{EDF}(I)| = 0$. Since EDF cannot serve any requests, then neither can OPT. So $|\text{OPT}(I)| = 0$, and the claim is satisfied.

Inductive step: For the inductive step, we thus consider $|\text{EDF}(I)| + T \ge 2$ and suppose the theorem holds for any instance I' where $|\text{EDF}(I')| + T < |\text{EDF}(I)| + T$.

We consider two cases depending upon the size of T compared to $|\text{EDF}(I)|$. Case 1: $T \le 2|\text{EDF}(I)| + 1$. In this straightforward case, because of the time limit, OPT can serve at most $2|\text{EDF}(I)| + 1$ requests, so the claim holds.
Case 2: $T \ge 2|\text{EDF}(I)| + 2$. Since $T - |\text{EDF}(I)| \ge |\text{EDF}(I)| + 2$, EDF$(I)$ must have $|\text{EDF}(I)| + 2$ or more empty drives, and thus at least two consecutive empty drives. As defined above, the earliest such occurrence is during the interval $[\tau, \tau + 2]$.

Summing the six labeled facts above and noting that $|R| = |X| + |Y| = X_0 + X_3 + X_4 + Y_1 + Y_2 + \ldots + Y_5$, we have

$$|R| - |R^*| = |C_1| + |C_3| + |C_0|. \tag{2}$$

We now define a sub-instance I' of the original instance I as follows. We remove all requests that EDF served by time τ, and we also remove any requests with deadline at most $\tau + 2$. The origin is at the point in the metric space where EDF(I) (EDF's schedule on the original instance) places the server at time $\tau + 1$; denote said point as m. The new time limit is $T - (\tau + 1)$, but to avoid confusion, we will view the time in I' as running from time $\tau + 1$ to time T.

Observe that by construction there are no servable requests in this sub-instance I' at time $\tau + 1$: any request with release time $\tau + 1$ or earlier and deadline greater than $\tau + 2$ would have already been served by EDF because otherwise EDF(I) would not have had two consecutive empty drives at time τ.

Let w^* denote the number of requests served by OPT(I) in $[0, \tau + 2]$. To give a lower bound on OPT's performance on the sub-instance (OPT(I')), we consider the subschedule of OPT(I) starting at time $\tau + 2$ continuing on OPT(I)'s path to time T. We let P denote this subschedule concatenated with a single drive from m to the front. The requests served by following P on I' number at least $|\text{OPT}(I)| - w^* - (|R| - |R^*|)$: this accounts for all of the requests that OPT(I) serves in time $[\tau + 2, T]$ (the $|\text{OPT}(I)| - w^*$) as well as those served by EDF(I) by time $\tau + 2$ with deadline $\tau + 2$ or later (collectively $|R|$), except for those OPT(I) had served by time $\tau + 2$ (which total $|R^*|$). Hence,

$$|\text{OPT}(I)| - w^* - (|R| - |R^*|) \leq |\text{OPT}(I')|. \tag{3}$$

By induction on the sub-instance I', since no request can be served in the first time unit, we have $|\text{OPT}(I')| \leq 2|\text{EDF}(I')|$. Observe that EDF$(I')$ has the same schedule as the suffix of EDF(I) starting from $\tau + 1$. Thus, since EDF(I) serves w requests by time τ, EDF(I') serves precisely $|\text{EDF}(I)| - w$ requests. Thus,

$$|\text{OPT}(I)| \leq w^* + (|R| - |R^*|) + 2|\text{EDF}(I)| - 2w. \tag{4}$$

To bound w^*, we let z be a binary indicator variable denoting whether or not OPT(I) serves a request at time $[\tau + 1, \tau + 2]$, with value 1 if it does. Thus, $w^* \leq (\tau + 1) + z - |E^*|$. Combined with Equations 4 and 2, we get

$$|\text{OPT}(I)| \leq \tau + 1 + z - |E^*| + (|C_1| + |C_3| + |C_0|) + 2|\text{EDF}(I)| - 2w. \tag{5}$$

We now consider how $|E^*|$ relates to other quantities, defining E' informally as the empty drives of OPT not appearing in any trace. Formally, let $E' = E^* - (E_1^* \cup E_2^*)$, since empty drives of OPT appear in traces of category C_1 or C_2 only. Using the observations that $E_1^* = |C_1|$ and $E_2^* = |C_2|$, we have

$$|E^*| = |E'| + |C_1| + |C_2|. \tag{6}$$

Substituting Equations 1 and 6 into Equation 5 gives:

$$|\text{OPT}(I)| \leq -|Q| + 1 + z - |E'| - |C_2| + |C_3| + |C_0| + 2|\text{EDF}(I)|. \tag{7}$$

Because each C_0, C_3 or C_4 trace has an element from Q and such requests are all distinct, $|Q| \geq |C_0| + |C_3| + |C_4|$. Combining that with Equation 7 gives

$$|\text{OPT}(I)| \leq 1 + z - |E'| - |C_2| - |C_4| + 2|\text{EDF}(I)|. \tag{8}$$

Observe that if $z = 1$, the OPT request at time $[\tau + 1, \tau + 2]$ begins a C_2 or C_4 trace, ensuring $|C_4| + |C_2| \geq z$, whether z is 1 or 0. Thus,

$$|\text{OPT}(I)| \leq 1 - |E'| + 2|\text{EDF}(I)|, \tag{9}$$

allowing us to conclude that $|\text{OPT}(I)| \leq 2|\text{EDF}(I)| + 1$.   □

If $\text{OPT}(I)$ did not serve a request in the first time unit we can remove the additive 1 term in the upper bound, getting a stronger formulation. Note that this case will always occur if there is no request whose source is the origin.

Theorem 3. *For any instance I of TDARPTW on which $\text{OPT}(I)$ does not serve a request during the time $[0, 1]$, $|\text{OPT}(I)| \leq 2|\text{EDF}(I)|$.*

Proof. In the straightforward case where where $T \leq 2|\text{EDF}(I)| + 1$, if OPT does not serve a request in the first time unit, then OPT can serve at most $2|\text{EDF}(I)|$, satisfying the claim.

For the other case, where $T \geq 2|\text{EDF}(I)| + 2$, we begin with Equation 9 from Theorem 2, $|\text{OPT}(I)| \leq 1 - |E'| + 2|\text{EDF}(I)|$, which holds for any instance of the problem, and now consider how it can be refined when $\text{OPT}(I)$ does not serve a request during $[0, 1]$. We consider whether $\text{EDF}(I)$ served a request during $[0, 1]$. If $\text{EDF}(I)$ also did not serve a request during $[0, 1]$ then $\text{OPT}(I)$'s empty drive at $[0, 1]$ does not appear in any trace (since according to the rules of traces, empty drives of EDF do not belong in any trace), but is in E', so $|E'| \geq 1$, and thus $|\text{OPT}(I)| \leq 2|\text{EDF}(I)|$, satisfying the claim.

If $\text{EDF}(I)$ does serve a request, call it r_1, during $[0, 1]$ we now show by induction on a smaller instance that the same conclusion is reached. This smaller instance $\mathcal{I}$ differs from the original input I in that its origin $\bar{o}$ is the destination of r_1, request r_1 is removed from the set of requests, and the time limit is $T - 1$. Note that $|\text{EDF}(\mathcal{I})| = |\text{EDF}(I)| - 1$.

Let P denote the path that proceeds from $\bar{o}$ to where $\text{OPT}(I)$ is at time 2 (call that location f) and then continues on $\text{OPT}(I)$'s path for the remaining $T - 2$ units of time. We will use P to serve as a proxy for OPT to bound the value of $\text{OPT}(\mathcal{I})$. Let z be an indicator variable denoting whether or not there is a servable request at time 2 from $\bar{o}$ to f. We lower bound the number of requests P served by considering that while P largely aligns with $\text{OPT}(I)$, if OPT served r_1, P will not because r_1 is no longer present in $\mathcal{I}$ and P may now make a different move during time $[1, 2]$. Accordingly, $|P| \geq |\text{OPT}(I)| - 2 + z$, or $|\text{OPT}(I)| \leq |P| + 2 - z$. By Theorem 2, $|P| \leq |\text{OPT}(\mathcal{I})| \leq 2(|\text{EDF}(I)| - 1) + 1$.

When $z = 1$, $|\text{OPT}(I)| \leq 2(|\text{EDF}(I)| - 1) + 1 + 2 - z = 2|\text{EDF}(I)|$. In the case $z = 0$, if $|P| = 2(|\text{EDF}(I)| - 1) + 1$, then P is an optimal path on $\mathcal{I}$. As such, the inductive hypothesis, $|\text{OPT}(\mathcal{I})| \leq 2(|\text{EDF}(\mathcal{I})|$, guarantees $|P| \leq 2(|\text{EDF}(I)| - 1)$. Thus, when $z = 0$, $|\text{OPT}(I)| \leq |P| + 2 - z \leq 2(|\text{EDF}(I)| - 1) + 2 - z = 2|\text{EDF}(I)|$.

Hence, the proof is complete.   □

From Theorems 1 and 3, we obtain the following corollary.

Corollary 1. *When no requests have their source at the origin, the approximation ratio of the* EDF *algorithm for TDARPTW is* 2 *(and this is tight).*

One may wonder if a better performance can be achieved if the server does not make an empty drive to serve an earlier-deadline request when there is a request available at its current location. Informally, the opportunistic EDF algorithm, denoted EDFO, determines the set of servable requests that begin where the server is located. If that set is nonempty, it chooses a request arbitrarily among those in the set with the earliest deadline; if that set is empty, it chooses a request as in EDF. Whenever t is incremented, it repeats this procedure. We show that the same upper bound holds for EDFO as for EDF. Moreover, this EDFO algorithm need not be better than EDF. Lemma 3 in the Appendix provides instances for which $|\text{OPT}| = |\text{EDF}|$ but $|\text{OPT}|/|\text{EDFO}|$ is arbitrarily close to 2.

Theorem 4. EDFO *is a 2-approximation for TDARPTW (and this is tight).*

Proof. Apply the proof of Theorem 2, replacing EDF with EDFO, except for the distinctions we now note. Few changes are needed since when EDF serves a request that is preceded by an empty drive, it chooses the earliest deadline among all servable requests. As such, all claims about the traces and almost all equations still hold. However, we need to consider the situation in which OPT serves a request in the first time slot, $[0, 1]$. In such a case, though EDF may not have served a request during $[0, 1]$, EDFO must; call this request $\bar{r}$.

Reviewing the definitions of various sets of requests, $\bar{r} \in Q$ as it is served by EDFO prior to time τ and not preceded by an empty drive. Accordingly, we have to adjust the proof of Theorem 2 from right after Equation 7 where we first consider the size of Q. Because $\bar{r}$ is not counted in C_0, C_3 or C_4 but is part of Q, we can state that for EDFO we have $|Q| \geq |C_0| + |C_3| + |C_4| + 1$. Combining this inequality with Equation 7 gives $|\text{OPT}(I)| \leq -|C_4| + z - |E'| - |C_2| + 2|\text{EDFO}(I)|$. Identically to the proof of Theorem 2, if $z = 1$, the OPT request at time $[\tau + 1, \tau + 2]$ begins a C_2 or C_4 trace, ensuring $|C_4| + |C_2| \geq z$, and hence that $|\text{OPT}(I)| \leq 2|\text{EDFO}(I)|$.

The matching lower bound may be found in the Appendix. □

4   Concluding Remarks

Preliminary experimental results indicate that while EDF may serve as few as half of the requests that OPT serves, that is rare; EDF often performs close to optimally and determines a schedule quickly [21].

A natural extension to TDARPTW would be to consider the problem in the online setting, where requests are not known until their release times. The work of [25] studies ODARPTW, which is the online form of TDARPTW, but without the overall time limit T. They show that an online algorithm that schedules requests greedily by "waiting time," which is a proxy for the deadline of the

request, is 3-competitive. For the online form of TDARPTW, preliminary investigations provide instances guaranteeing that the online form of EDF has a competitive ratio no better than 3; our conjecture is that the online form of EDF is in fact 3-competitive for the online form of TDARPTW, consistent with [25]. However, if requests were known one time unit prior to their release, perhaps akin to how users call-ahead for ride services, the ratio of 2 from the offline setting holds. This is because our offline EDF algorithm uses information about a request only one time unit before it serves the request.

References

1. Anthony, B.M., Birnbaum, R., Boyd, S., Christman, A., Chung, C., Davis, P., Dhimar, J., Yuen, D.S.: Maximizing the number of rides served for dial-a-ride. In: 19th ATMOS Symposium. vol. 75, pp. 11:1–11:15 (2019)
2. Anthony, B.M., Christman, A.D., Chung, C., Yuen, D.: Serving rides of equal importance for time-limited dial-a-ride. In: MOTOR. pp. 35–50 (2021)
3. Baligács, J., Disser, Y., Mosis, N., Weckbecker, D.: An improved algorithm for open online dial-a-ride. In: Approximation and Online Algorithms. pp. 154–171. Springer International Publishing, Cham (2022)
4. Bansal, N., Blum, A., Chawla, S., Meyerson, A.: Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In: 36th Annual ACM Symposium on Theory of Computing. pp. 166–174 (2004)
5. 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)
6. Birx, A., Disser, Y., Schewior, K.: Improved bounds for open online dial-a-ride on the line. Algorithmica **85**(5), 1372–1414 (2023)
7. Charikar, M., Raghavachari, B.: The finite capacity dial-a-ride problem. In: 39th Annual Symposium on Foundations of Computer Science. pp. 458–467 (1998)
8. Chen, S., He, T., Wong, H.Y.S., Lee, K.W., Tong, L.: Secondary job scheduling in the cloud with deadlines. In: IEEE IPDPS. pp. 1009–1016 (2011)
9. Chetto, H., Chetto, M.: Some results of the earliest deadline scheduling algorithm. IEEE Transactions on Software Engineering **15**(10), 1261–1269 (1989)
10. Christman, A., Chung, C., Jaczko, N., Milan, M., Vasilchenko, A., Westvold, S.: Revenue maximization in online dial-a-ride. In: 17th ATMOS Workshop. vol. 59, pp. 1:1–1:15. Dagstuhl, Germany (2017)
11. Christman, A.D., Chung, C., Jaczko, N., Li, T., Westvold, S., Xu, X., Yuen, D.: Improved Bounds for Revenue Maximization in Time-Limited Online Dial-a-Ride. SN Operations Research Forum **2**(3), 1–38 (2021)
12. Cordeau, J.F., Laporte, G.: The dial-a-ride problem: Models and algorithms. Annals of Operations Research **153**(1), 29–46 (Sep 2007)
13. Dertouzos, M.L.: Control robotics: The procedural control of physical processes. In: Proceedings IFIP Congress, 1974 (1974)
14. Dertouzos, M.L., Mok, A.K.: Multiprocessor online scheduling of hard-real-time tasks. IEEE Transactions on Software Engineering **15**(12), 1497–1506 (1989)
15. Desrosiers, J., Dumas, Y., Soumis, F.: A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. American Journal of Mathematical and Management Sciences **6**(3-4), 301–325 (1986)

16. Frederickson, G.N., Hecht, M.S., Kim, C.E.: Approximation algorithms for some routing problems. In: 17th Annual Symposium on Foundations of Computer Science. pp. 216–227 (1976)
17. Ho, S.C., Szeto, W., Kuo, Y.H., Leung, J.M., Petering, M., Tou, T.W.: A survey of dial-a-ride problems: Literature review and recent developments. Transportation Research Part B: Methodological **111**, 395–421 (2018)
18. Jaw, J.J., Odoni, A.R., Psaraftis, H.N., Wilson, N.H.: A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. Transportation Research Part B: Methodological **20**(3), 243–257 (1986)
19. Leung, J.Y.: Handbook of scheduling: Algorithms, models, and performance analysis. CRC Press (2004)
20. Madsen, O.B., Ravn, H.F., Rygaard, J.M.: A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. Annals of Operations Research **60**, 193–208 (1995)
21. Medina, A., Anthony, B.M.: Evaluating an earliest deadline first algorithm for a dial-a-ride problem [poster]. Tapia Celebration of Diversity in Computing (2023)
22. Molenbruch, Y., Braekers, K., Caris, A.: Typology and literature review for dial-a-ride problems. Annals of Operations Research **259**(1), 295–325 (2017)
23. Solomon, M.M., Desrosiers, J.: Survey paper—time window constrained routing and scheduling problems. Transportation Science **22**(1), 1–13 (1988)
24. Stankovic, J.A., Spuri, M., Ramamritham, K., Buttazzo, G.C.: Fundamentals of EDF Scheduling, pp. 27–65. Springer US, Boston, MA (1998)
25. Yi, F., Song, Y., Xin, C., Walter, I., Kai, K.W.: Online dial-a-ride problem with unequal-length time-windows. In: 2009 International Conference on Management and Service Science. pp. 1–5 (2009)
26. Yi, F., Tian, L.: On the online dial-a-ride problem with time-windows. In: Algorithmic Applications in Management. pp. 85–94. Springer Berlin Heidelberg (2005)
27. Yi, F., Xu, Y., Xin, C.: Online dial-a-ride problem with time-windows under a restricted information model. In: Alg. Aspects Inf. Management. pp. 22–31 (2006)

Appendix

This Appendix contains some additional figures and proofs that the conference version of this paper did not include due to space limitations.

		Starting Drive Types	
		$Y \subset R$ (EDF)	R^* (OPT)
Termination	E^* (OPT)	C_1	C_2
Drive Types	$X \subset Q$ (EDF)	C_3	C_4

Table 1. Categorizing maximal traces that alternate between OPT and EDF. See also Figure 3.

Proof of Lemma 1

Lemma 1. *In a maximal trace, every request served by a drive of* OPT *must also be served by a drive of* EDF*, and these drives never occur past* $\tau + 2$.

Proof. Let r be a request served by OPT in a maximal trace. If $r \in R^*$, then $r \in R$ and we are done. So let $r \notin R^*$. This trace must have started either by EDF serving a request from Y or OPT serving a request from R^*. In the latter case, since the trace must continue with EDF serving a request from R (by Rule 2), and if the request from R is in X, the trace would terminate before reaching r, the request from R must be in Y. So we assume that the EDF drive serving a request in Y appears in the trace sequence before the OPT drive serving r. Let r_0 be the request in Y (served by EDF) that is nearest in the trace preceding r, so that all requests in between r_0 and r are not in R (so their deadlines are before $\tau + 2$ and hence, they are not in R^* either). We now prove the following stronger claim where r need not be served by OPT but its time window must satisfy certain constraints.

Claim: Let $n \geq 0$ be an integer. Let a trace contain the following sequence.

- EDF serves $r_0 \in Y$ at time t_0.
- For $1 \leq i \leq n$, OPT serves r_i at time t_{i-1} and EDF serves r_i at time t_i, with $r_i \notin R$ and $r_i \notin Q$.

Let r be a request with release time $\rho \leq t_n$ and deadline $d > t_n$. (Note that unless r was already served by EDF before time t_n, then r could be served by EDF at time t_n.) Then r must be served by EDF.

Proof of Claim: We proceed by induction on n. Consider the base case of $n = 0$. If, for contradiction, r is not served by EDF, then because EDF chose to serve r_0, then the deadline d of r is at least that of r_0, and hence $d \geq \tau + 2$. Since EDF had two empty drives during $[\tau, \tau + 2]$, if r was available to be served at time τ, EDF would have served it. Since EDF served nothing at time τ or $\tau + 1$, r must have been served prior to time τ. Thus r is served by EDF in this base case.

Now fix $n \geq 1$ and assume the claim is true for smaller values of n. Denote the deadline of r_i by d_i ($0 \leq i \leq n$). Note for $1 \leq i \leq n$, we have $d_i > t_i$ and $d_i > t_{i-1}$. Note because EDF has served at least $n + 1 \geq 2$ requests, then all time values $t_0, \ldots, t_n$ are distinct. Assume by way of contradiction that EDF does not serve r. Because EDF chose to serve r_n at time t_n instead of r, then $d \geq d_n$. Consider two cases by comparing t_n and t_{n-1}.

Case $t_n < t_{n-1}$. Here EDF served r_{n-1} at time t_{n-1}. We have $d \geq d_n > t_{n-1}$ and since $\rho \leq t_n < t_{n-1}$, then by induction applied to $n - 1$ in place of n, EDF must serve r and we are done in this case.

Case $t_{n-1} < t_n$. Since both r_n and r_{n-1} could have been served by EDF at time t_{n-1} and EDF served r_{n-1}, we must have $d_{n-1} \leq d_n$. We now prove a useful fact that if for some $j \geq 2$,

$$t_i < t_n \text{ for all } n - j \leq i \leq n - 1, \text{and} \tag{10}$$

$$d_i \leq d \text{ for all } n - j + 1 \leq i \leq n, \tag{11}$$

then we have $d_{n-j} \leq d$. If $t_{n-j} < t_{n-j+1}$, then since EDF served r_{n-j} instead of r_{n-j+1} at time t_{n-j} (both requests could have been served by EDF at that time), then $d_{n-j} \leq d_{n-j+1} \leq d$ as desired. If $t_{n-j} > t_{n-j+1}$, then looking at

the sequence $t_{n-j+1}, t_{n-j+2}, \ldots, t_n$ (where the sequence increases and possibly decreases at times), because $t_{n-j} \in [t_{n-j+1}, t_n]$, then $t_{n-j} \in [t_k, t_{k+1}]$ for some $n - j + 1 \leq k \leq n - 1$. Now the request r_{k+1} was served by OPT at time t_k and served by EDF at time t_{k+1}. Hence EDF chose to serve r_{n-j} instead of r_{k+1} at time t_{n-j}. Therefore $d_{n-j} \leq d_{k+1} \leq d$. This completes the proof of the useful fact.

If all $t_i < t_n$ for $0 \leq i < n$, then using $d_{n-1} \leq d$ and repeated applications of the useful fact, we would deduce that $d_i \leq d$ for all $0 \leq i \leq n-1$. But $d_0 > \tau > d$ would be a contradiction with $d_0 \leq d$. Let j be the integer for which $t_{n-j} > t_n$ is minimized. Then $t_i < t_n$ for $n - j + 1 \leq i \leq n - 1$ and the useful fact implies $d_{n-j} \leq d$ This implies $t_{n-j} < d$. Along with $t_{n-j} > t_n \geq \rho$, this implies r is could be served by EDF at time t_{n-j}. Since $n - j < n$, then by induction, EDF must serve r. This contradiction completes the induction and the proof of the claim. The lemma follows easily from the claim.   □
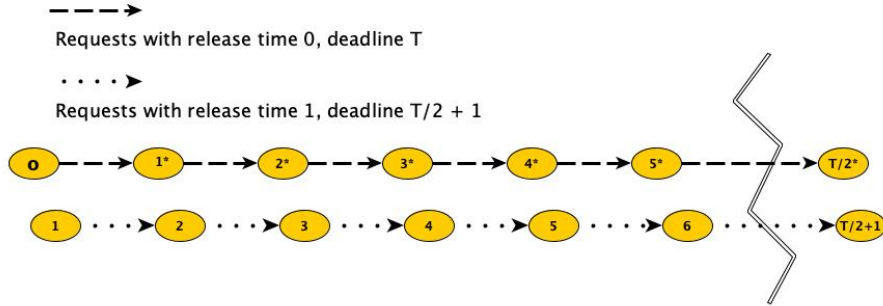

Fig. 4. The server starts at the origin o. Let T be even. All requests are represented by arrows, either dashed or dotted. The top row consists of $T/2$ requests, each with release time 0 and deadline T. The bottom row illustrates $T/2$ requests, each with release time 1 and deadline $T/2 + 1$.

We now present an instance where EDFO serves half of OPT, but EDF could be optimal. Observe that in Figure 4 one possible solution achieved by EDF is serving the bottom row of requests in numerical order (in time $T/2 + 1$), then proceeding to serve as many of the top row of requests as it can, in numerical order, for a total of $T - 2$ requests served; time units $[0, 1]$ and $[T/2+1, T/2+2]$ are empty drives. Thus $|\text{EDF}| = T - 2$, while EDFO serves only $T/2$ requests (see Lemma 3 for what is served by OPT and EDFO). We note that while OPT and EDFO will always do what is described, EDF may not make optimal decisions; it would pick a request in the first row to serve, but it wouldn't necessarily traverse the requests in the first row in the same order as OPT.

Lemma 3. *If, for all instances I of TDARPTW, $|\text{OPT}(I)|/|\text{EDFO}(I)| \leq \rho$ for some ratio ρ, then $\rho \geq 2$.*

Proof. Consider the instance shown in Figure 4. Because the origin o is at the source of a request, EDFO will serve it immediately, and then continue to serve all requests on the top row, completing the final request of the top row at time $T/2$. With the required one unit empty drive before any other request can be served, it will be unable to serve any additional requests as the bottom row's deadlines are all $T/2 + 1$. Hence $|\text{EDFO}| = T/2$.

On the other hand, OPT serves the bottom row of requests in numerical order (in time $T/2 + 1$), then proceeds to serve as many of the top row of requests as it can, in numerical order, for a total of $T - 2$ requests served; time units $[0, 1]$ and $[T/2 + 1, T/2 + 2]$ are empty drives. There is no way to serve more requests than this, since the travel time to get to the bottom row is necessary, and if any requests in the top row were served before proceeding to the bottom row, there would be insufficient time to serve all bottom row requests prior to their deadline. Thus $|\text{OPT}| = T - 2$.

As T grows, the ratio $|\text{OPT}|/|\text{EDFO}|$ approaches 2.   □