Serving rides of equal importance for time-limited Dial-a-Ride

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

[1] Southwestern University, Georgetown, Texas, USA anthonyb@southwestern.edu
[2] Middlebury College, Middlebury, Vermont, USA achristman@middlebury.edu
[3] Connecticut College, New London, Connecticut, USA cchung@conncoll.edu
[4] 1507 Punawainui St., Kapolei, Hawaii, USA yuen888@hawaii.edu

Abstract. We consider a variant of the offline Dial-a-Ride problem with a single server where each request has a source, destination, and a prize earned for serving it. The goal for the server is to serve requests within a given time limit so as to maximize the total prize money. We consider the variant where prize amounts are uniform which is equivalent to maximizing the number of requests served. This setting is applicable when all rides may have equal importance such as paratransit services. We first prove that no polynomial-time algorithm can be guaranteed to serve the optimal number of requests, even when the time limit for the algorithm is augmented by any constant factor $c \geq 1$. We also show that if $\lambda = t_{max}/t_{min}$, where t_{max} and t_{min} denote the largest and smallest edge weights in the graph, the approximation ratio for a reasonable class of algorithms for this problem is unbounded, unless λ is bounded. We then show that the SEGMENTED BEST PATH (SBP) algorithm from [7] is a 4-approximation. We then present our main result, an algorithm, k-Sequence, that repeatedly serves the fastest set of k remaining requests, and provide upper and lower bounds on its performance. We show k-Sequence has approximation ratio at most $2+\lceil\lambda\rceil/k$ and at least $1+\lambda/k$ and that $1+\lambda/k$ is tight when $1+\lambda/k \geq k$. Thus, for the case of $k = 1$, i.e., when the algorithm repeatedly serves the quickest request, it has approximation ratio $1 + \lambda$, which is tight for all λ. We also show that even as k grows beyond the size of λ, the ratio never improves below $9/7$.

1 Introduction

In the Dial-a-Ride Problem (DARP) one or more servers must schedule a collection of pickup and delivery requests, or rides. Each request specifies the pickup location (or *source*) and the delivery location (or *destination*). In some DARP variants the requests may be restricted so that they must be served within a specified time window, they may have weights associated with them, or details about them may be known only when they become available. For most variations the goal is to find a schedule that will allow the server(s) to serve requests within the constraints, while meeting a specified objective. Much of the motivation for

DARP arises from the numerous practical applications of the transport of both people and goods, including delivery services, ambulances, ride-sharing services, and paratransit services. For a comprehensive overview of DARP please refer to the surveys *The dial-a-ride problem: models and algorithms* [9] and *Typology and literature review for dial-a-ride problems* [13].

In this work we study offline DARP on weighted graphs with a single server where each request has a source, destination, and prize amount. The prize amount may represent the importance of serving the request in settings such as courier services. In more time-sensitive settings such as ambulance routing, the prize may represent the urgency of a request. In profit-based settings, such as taxi and ride-sharing services, a request's prize amount may represent the revenue earned from serving the request. The server has a specified deadline after which no more requests may be served, and the goal is to find a schedule of requests to serve within the deadline that maximizes the total prize money. We study the variant where prizes are uniform so the goal is equivalent to maximizing the number of requests served within the deadline. This variant is useful for settings where all requests have equal importance, such as nonprofit transportation services for elderly and disabled passengers and courier services where deliveries are not prioritized. For the remainder of this paper, we will refer to this time-limited variant with the objective of maximizing the number requests served as TDARP.

One related problem is the Prize Collecting Traveling Salesperson Problem (PCTSP) where the server earns a prize for every location it visits and a penalty for every location it misses, and the goal is to collect a specified amount of prize money while minimizing travel costs and penalties. PCTSP was introduced by Balas [3] but the first approximation algorithm, with ratio 2.5, was given by Bienstock et al. [4]. Later, Goemans and Williamson [11] developed a primal-dual algorithm to obtain a 2-approximation. Building off of the work in [11], Archer et al. [2] improved the ratio to $2 - \epsilon$, a significant result as the barrier of 2 was thought to be unbreakable. More recently, Paul et al. [14, 15] studied a special case of our problem; namely, the *budgeted* variant of PCTSP where the goal is to find a tour that maximizes the number of nodes visited given a bound on the cost of the tour. They present a 2-approximation when the graph is not required to be complete and the tour may visit nodes more than once.

Blum et al. [5] presented the first constant-factor approximation algorithm for a special case of the problem we consider; namely, the *Orienteering Problem* where the input is a weighted graph with rewards on nodes and the goal is to find a path that starts at a specified origin and maximizes the total reward collected, subject to a limit on the path length. Our problem is a generalization of this problem – while the Orienteering Problem has as input a set of points/cities to visit, our problem has a set of requests, each with two distinct points to be visited: a source and a destination.

To our knowledge, despite its relevance to modern-day transportation systems, aside from the work in [1] the request-maximizing time-limited version of DARP we investigate in this paper has not been previously studied in the offline

setting. In [1] we presented a 3/2-approximation algorithm, TWOCHAIN, for the more restrictive uniform edge weight version of the problem.

Our results In Section 2 we begin by establishing some impossibility results. In Section 2.1 we prove that no polynomial-time algorithm can be guaranteed to serve as many requests as the optimal schedule, even when the time limit T for the algorithm is augmented by c for any constant $c \geq 1$. We also show that if $\lambda = t_{max}/t_{min}$, where t_{max} and t_{min} denote the largest and smallest edge weights in the graph, the approximation ratio for a reasonable class of TDARP algorithms is unbounded, unless λ is bounded. In Section 2.2 we revisit the SEGMENTED BEST PATH (SBP) algorithm that was proposed in [7] for TDARP in the online setting. We show that SBP in the offline setting is a 4-approximation, and we also show this is a tight bound.

In Section 3 we present *k-Sequence* (k-SEQ), a family of algorithms parameterized by k, for TDARP on weighted graphs. Informally, the k-SEQ algorithm repeatedly serves the fastest set of k remaining requests where a determination of *fastest* is made by considering both the time to serve the requests and any travel time necessary to serve those requests. Naturally, k is a positive integer. Our approximation ratio depends on λ, a property of the graph, similar to the graph-property dependencies in [6, 10]. In many real-world settings, λ may be viewed as a constant [8, 12, 16]. We prove that k-SEQ has approximation ratio $2+\lceil\lambda\rceil/k$. In Section 3.1 we show that when $1+\lambda/k \geq k$, the approximation ratio for k-SEQ improves to $1+\lambda/k$. Thus, for the case of $k = 1$, i.e., the polynomial-time algorithm which repeatedly serves the quickest request, the approximation ratio is $1+\lambda$ and this is tight. Finally, in Section 4, we show that k-SEQ has approximation ratio at least $1+\lambda/k$, which matches the upper bound for when $1+\lambda/k \geq k$. We also show that the algorithm has a lower bound of $9/7$ for $k > \lambda$.

We summarize our results on the approximation ratio for k-SEQ, for particular λ and k, as follows.

1. When $\lambda \geq k(k-1)$, or equivalently $1+\lambda/k \geq k$, the ratio is $1+\lambda/k$, and this is tight. So when $k = 1$ (for any λ), the ratio is $1+\lambda$, and this is tight.
2. When $k \leq \lambda < k(k-1)$, then the ratio is in the interval $[1+\lambda/k, 2+\lceil\lambda\rceil/k]$.
3. When $\lambda < k$, the ratio is in the interval $[\max\{9/7, 1+\lambda/k\}, 2+\lceil\lambda\rceil/k]$.

2 Preliminaries

We formally define TDARP as follows. The input is an undirected complete graph $G = (V, E)$ where V is the set of vertices (or nodes) and $E = \{(u, v) : u, v \in V, u \neq v\}$ is the set of edges. For every edge $(u, v) \in E$, there is a distance $dist(u, v) > 0$, which represents the amount of time it takes to traverse (u, v).[†] We also note that the input can be regarded as a metric space if the weights

[†]We note that any simple, undirected, connected, weighted graph is allowed as input, with the simple pre-processing step of adding an edge wherever one is not present whose distance is the length of the shortest path between its two endpoints.

on the edges are expected to satisfy the triangle-inequality. Indeed, all of our results apply to both complete graphs as well as metric spaces.

One node in the graph, o, is designated as the origin and is where the server is initially located (i.e. at time 0). The input also includes a time limit T and a set of requests, S, that is issued to the server. Each request in S can be considered as simply a pair (s, d) where s is the source node or starting point of the request, and d is the destination node. The output is a schedule of requests, i.e. a set of requests and the time at which to serve each. To serve a request, the server must move from its current location x to s, then from s to d, and remain at d until it is ready to move again. The total time for serving the request is $dist(x, s) + dist(s, d)$, where $dist(x, s) = 0$ if $x = s$.

Every movement of the server can be characterized as either an *empty drive* which is simply a repositioning move along an edge but not serving a request, or a *service drive* in which a request is being served while the server moves. We let $driveTime(C)$ denote the minimum total time for the server to travel from its current location and serve the collection of requests $C \subseteq S$, where the minimum is taken over all permutations of the requests in C.

We use $|\textsc{alg}(I)|$ to denote the number of requests served by an algorithm ALG on an instance I of TDARP and we drop the I when the instance is clear from context. Similarly we use $|\textsc{opt}(I)|$ for the number of requests served by the optimal solution OPT on instance I.

2.1 Impossibility results

In this section we present two impossibility results. The first is an inapproximability result. The second demonstrates that any algorithm of a class of algorithms that k-SEQ belongs to will have unbounded approximation ratio, unless λ is bounded. Accordingly, this provides some justification for the presence of the parameter λ in our main results.

2.1.1 c-Time inapproximability

We prove that, unless P=NP, no polynomial-time algorithm can be guaranteed to serve as many requests as the optimal schedule, even when the time limit T for the algorithm is augmented by any constant factor. Let $I = (G, S, T)$ denote an instance of TDARP, where G is the input graph, S is the set of requests, and T is the time limit. We define ALG to be a *ρ-time-approximation* if ALG serves at least as many requests as OPT on the instance $(G, S, \rho T)$. The proof idea is to show that a polynomial-time *c-time-approximation* to TDARP yields a polynomial-time decider for the directed Hamiltonian path problem. Please see Appendix 6.2 for the proof.

Theorem 1. *If $P \neq NP$, then there is no polynomial-time c-time-approximation to TDARP for any constant $c \geq 1$.*

2.1.2 Inductive stateless greedy algorithms

Recall that $\lambda = t_{max}/t_{min}$, where t_{max} and t_{min} denote the largest and smallest edge weights in the graph, respectively. We now show that if a deterministic algorithm satisfies certain properties, then it cannot have a bounded approximation ratio, unless λ is bounded. Consider the following three properties of an algorithm. (Note these are abbreviated summaries of each property; please see Appendix 6.3 for more detailed definitions and the proof of Theorem 2).

1. *Inductive.* The algorithm chooses paths to take in stages.
2. *Stateless.* In each stage the algorithm does not use state information from a previous stage.
3. *Greedy.* The algorithm makes decisions by optimizing an objective function at each stage, where the function takes as input a set of possible paths to choose from and outputs a chosen path.

Theorem 2. *Let M be a constant and let* ALG *be a deterministic inductive stateless greedy algorithm such that the algorithm considers only candidate paths with at most M edges. If λ is not bounded, then* ALG *has an unbounded approximation ratio.*

2.2 The Segmented Best Path (SBP) algorithm

Before we present our main results, we will now analyze an algorithm that is based on the previously-studied SEGMENTED BEST PATH (SBP) algorithm from [7], which was proposed for the *online* variant of DARP with non-uniform prize amounts. Specifically, we adapt SBP to apply in our offline setting with uniform prize amounts. Since our problem assumes uniform prizes, we unsurprisingly have a tighter upper bound than the bound of [7], but we show that the lower bound carries over. We note that Theorem 2 does not apply to SBP because there is no constant that bounds the number of edges in the paths considered by SBP in each iteration.

Algorithm 1: SEGMENTED BEST PATH (SBP) Algorithm as adapted from [7]. Input: origin o, time limit $T > 0$, a complete graph G (see footnote † in Section 2) with $T \geq 2t_{max}$, and a set of requests S given as source-destination pairs.

Let $t_1, t_2, \ldots t_f$ denote time segments of length T/f ending at times $T/f, 2T/f, \ldots, T$, respectively, where $f = 2\lfloor T/(2t_{max})\rfloor$.
Let $i = 1$.
while $i < f$ and there are still unserved requests **do**
    At the start of t_i, find the *max-cardinality-sequence*, $\mathcal{R}$.
    Move to the source location of the first request in $\mathcal{R}$.
    At the start of t_{i+1}, serve the requests in $\mathcal{R}$.
    Let $i = i + 2$.
end while

As described in Algorithm 1, the offline version of SBP starts by splitting the total time T into $f \geq 2$ time segments each of length T/f where f is the largest even integer such that $t_{max} \leq T/f$, which ensures any move, including one serving a request, can be completed entirely in a single segment. At the start of a time segment, the server determines the *max-cardinality-sequence*, $\mathcal{R}$, i.e. the maximum length sequence of requests that can be served within one time segment, and moves to the source of the first request in this set. During the next time segment, it serves the requests in this set. It continues this way, alternating between determining and moving to the source of the first request in $\mathcal{R}$ during one time segment, and serving the requests in $\mathcal{R}$ in the next time segment.[‡]

Finding the max-cardinality-sequence may require enumeration of all possible sequences of unserved requests which takes time exponential in the number of unserved requests. However, in many real world settings, the number of requests will be small relative to the input size and in settings where T/f is small, the runtime is further minimized. Therefore it should be feasible to execute the algorithm efficiently in many real world settings.

Let $\text{OPT}(S, T, o)$ and $\text{SBP}(S, T, o)$ denote the schedules returned by OPT and SBP, respectively, on the instance (S, T, o).

Theorem 3. SBP *is a 4-approximation i.e.,* $|\text{OPT}(S, T, o)| \leq 4|\text{SBP}(S, T, o)|$ *for any instance* (S, T, o) *of TDARP, and this is tight.*

Proof. We first note that the lower bound instance of [7], in which SBP earns total prize money of no more than OPT/4 in the online setting, also applies to this offline setting with uniform prizes, since in that instance prize amounts are uniform and no requests are released after time 0. (For the full proof of the lower bound, please see Theorem 8 in the Appendix).

For the upper bound, consider a schedule OPT_2, which is identical to OPT except it is allowed one extra empty drive at the start that does not add to the overall time taken by the algorithm. More formally, if the first move in OPT is from o to some node n_1, then OPT_2 may have an additional (non-time-consuming) move at the start such that its first move is from o to some other node n_1' and its second move is from n_1' to n_1. Since OPT_2 is allowed one additional empty drive, we know $|\text{OPT}_2(S, T, o)| \geq |\text{OPT}(S, T, o)|$. We claim that $|\text{SBP}(S, T, o)| \geq |\text{OPT}_2(S, T, o)|/4$, which implies that $|\text{SBP}(S, T, o)| \geq |\text{OPT}(S, T, o)|/4$.

We proceed by strong induction on the number of time windows $w = f/2$ where a time window is two consecutive time segments. For the base case let Q and R denote the set of requests served by OPT_2 and SBP, respectively, in the first time window and let q and r denote their respective cardinalities. Recall the greedy nature of SBP which serves requests during every other time segment. If $q = 1$, since $f \geq 2$, SBP can serve the one request in Q within the two time segments so $r = q$. If $q > 1$, then if q is even, $r \geq q/2$ since splitting the window

[‡]Note that the algorithm need not take a full time segment to move from one set of requests to another, but it is specified this way for convenience of analysis. Excluding this buffer time in the algorithm specification does not improve its approximation ratio since one can construct an instance where each move requires the full time segment.

in half leaves at least half of the requests in one of the two time segments, and if q is odd then $r \geq (q-1)/2$.

So if $w = 1$, in all three cases, we have $r \geq q/4$, completing the base case.

For the inductive step, let P denote the path traversed by OPT_2, let $p = |\text{OPT}_2| \geq |\text{OPT}|$ denote the number of requests served in P, and let u denote the first node OPT_2 visits after the end of the first time window. Consider the subpath, P', of P that starts at u. Since P may contain a request that straddles the first two windows, P' contains at least $p-(q+1)$ requests. Let s_1 denote the last node SBP visits before the start of the second time window. After the first window, SBP is left with a smaller instance of the problem $(S_{new}, T_{new}, o_{new})$ where $S_{new} = S - R$, $T_{new} = T - T/f$, and $o_{new} = s_1$. So P' contains at least $p-(q+1)-r$ requests from this smaller instance and OPT_2 on $(S_{new}, T_{new}, o_{new})$ can move from o_{new} to u and serve these requests. By induction, on the smaller instance SBP will serve at least $(p-q-1-r)/4$. Thus

$$\begin{aligned} |\text{SBP}(S,T,o)| = r + |\text{SBP}(S_{new}, T_{new}, o_{new})| &\geq r + (p-q-1-r)/4 \\ &\geq p/4 + (-q-1+3r)/4. \quad (1) \end{aligned}$$

There are three cases for q and r.

1. Case: $q \geq 5$. Then since $r \geq (q-1)/2$, from (1) we have: $|\text{SBP}(S,T,o)| \geq p/4 + (-q-1+3(q-1)/2)/4 \geq p/4 + (q/2-5/2)/4 \geq p/4$.
2. Case: $q \leq 4$ and $r \geq 2$. From (1) we have: $|\text{SBP}(S,T,o)| \geq p/4 + (-4-1+6)/4 \geq p/4$.
3. Case: $q \leq 4$ and $r \leq 1$. If $r = 0$, then $|\text{SBP}(S,T,o)| = |\text{OPT}_2(S,T,o)| = 0$, so the theorem is trivially true, therefore, we assume $r = 1$. We first show by contradiction that every time window in OPT_2's schedule has fewer than 4 requests that end in that window. Suppose there is a window i in OPT_2's schedule that has 4 or more requests that end in window i. Then there are at least 3 requests that start and end in window i. This implies that at least one time segment of window i contains at least 2 requests which, by the greediness of SBP, implies $r \geq 2$, which is a contradiction since we are in the case where $r = 1$. Let w' denote the number of windows in which OPT_2 serves at least 1 request. We have $|\text{OPT}_2(S,T,o)| < 4w'$ and $|\text{SBP}(S,T,o)| \geq \min(w, |S|) \geq w'$, so $|\text{SBP}(S,T,o)| \geq |\text{OPT}_2(S,T,o)|/4$. ■

3 k-Sequence algorithm and upper bound

We now present k-Sequence (k-SEQ), our family of algorithms parameterized by k, for TDARP (see Algorithm 2). For any fixed k, the algorithm repeatedly serves the fastest set of k remaining requests where a determination of *fastest* is made by considering both the time to serve the requests and any travel time necessary to serve those requests. If there are fewer than k requests remaining, the algorithm exhaustively determines how to serve all remaining requests optimally. If the remaining time is insufficient to serve any collection of k requests, the algorithm likewise serves the largest set of requests that can be served within the remaining

time. We suggest that when using the algorithm in practice, k can be set as a small constant. The algorithm will run in time $O(|S|^{k+1})$ where S is the set of requests, as each of the at most $|S|$ iterations may require time $O(|S|^k)$.

Algorithm 2: Algorithm k-Sequence (k-SEQ). Input: origin o, time limit $T > 0$, a complete graph G (see footnote † in Section 2), and a set of requests S given as source-destination pairs.

Set $t := T$.
while there are at least k unserved requests remaining **do**
  Let C be the collection of k requests with fastest $driveTime(C)$, where $driveTime(C)$ denotes the minimum total time to serve C.
  if $t \geq driveTime(C)$ **then**
    Serve C, update $t := t - driveTime(C)$, and update $S = S - C$.
  else
    Exit **while** loop.
  end if
end while
Find the largest $x \leq k-1$ s.t. $driveTime(C') \leq t$ for some C' with $|C'| = x$.
If $|C'| \neq 0$, serve C'.

Theorem 4. *k-SEQ is a $(2 + \lceil \lambda \rceil / k)$-approximation for TDARP.*

Proof. First, note that without loss of generality, we may assume that it is possible to serve k requests during the allotted time T. If there was insufficient time to serve any collection of k requests, then k-SEQ will serve the largest set of requests that can be served within time T, which is thus optimal. If there are fewer than k requests available, k-SEQ will serve all available requests, again achieving an optimal solution.

We now proceed with a proof by induction on an instance in which at least k requests can be served in time T. For the base case, we have $\lfloor T/t_{min} \rfloor = 0$, so $T < t_{min}$ and thus k-SEQ and OPT both serve 0 requests, so we are done. For the inductive case, let $\lfloor T/t_{min} \rfloor = d \geq 1$. Suppose by induction that the theorem is true whenever $\lfloor T/t_{min} \rfloor < d$.

Let $s = o$ be the start location. k-SEQ starts by serving exactly k requests in time T_1, ending at a location we refer to as s_1. Let OPT serve m requests in total. Note that since T_1 is, by construction of k-SEQ, the time required to serve the fastest k requests, OPT serves at most k requests during the initial T_1 time.

Let y' be the location on the OPT path at time T_1, noting that y' need not be at a node. Then define y to be y' if y' is a node, or the next node on OPT's path after y' otherwise.

To develop our inductive argument, we will now create a new instance with new start location s_1, time $T_{new} = T - T_1$, and the k requests that were served by k-SEQ removed from S, leaving us with S_{new}.

We consider P, a feasible path for this new instance (see Figure 1). This path P starts at s_1, proceeds to y, and then traverses as much as it can of the remainder of the original OPT path from y in the remaining time, that is $T - T_1 - dist(s_1, y) \geq T - T_1 - t_{max}$.[§] Since such a path P is feasible, OPT's path must contain at least as many requests as P. Observe that the segment of the OPT path that P uses from y onward has a distance of at most $T - T_1$.

Since P has time at least $T - T_1 - t_{max}$ left when at y, then P misses at most $(T - T_1) - (T - T_1 - t_{max}) = t_{max}$ time of the tail of the original OPT path in addition to missing the initial T_1 time of the head of the original OPT path. Thus, P misses at most k requests from the head of the original OPT path and at most $\lceil \lambda \rceil$ from the tail of the original OPT path, ensuring that P serves at least $m - k - \lceil \lambda \rceil$ requests from the original instance. Since the new instance had k requests from the original instance removed, we can now say that P serves at least $m - 2k - \lceil \lambda \rceil$ requests from the new instance. Naturally, OPT must also serve at least $m - 2k - \lceil \lambda \rceil$ requests on the new instance.

Note t_{min} and t_{max}, and therefore λ, remain the same in the new instance. The allotted time for the new instance is $T_{new} = T - T_1 \leq T - t_{min}$, giving $\lfloor (T - T_1)/t_{min} \rfloor \leq \lfloor (T - t_{min})/t_{min} \rfloor = \lfloor T/t_{min} \rfloor - 1$. Hence by induction, the theorem is true for this new instance. In other words, the number of requests served by k-SEQ on the new instance is at least $k/(2k + \lceil \lambda \rceil)$ times the number of requests served by OPT on the new instance. Thus, $|k\text{-SEQ}(S, T, o)| = k + |k\text{-SEQ}(S_{new}, T_{new}, s_1)| \geq k + k/(2k + \lceil \lambda \rceil)(m - 2k - \lceil \lambda \rceil) \geq k + km/(2k + \lceil \lambda \rceil) + k(-2k - \lceil \lambda \rceil)/(2k + \lceil \lambda \rceil) = km/(2k + \lceil \lambda \rceil)$, completing the induction. ■

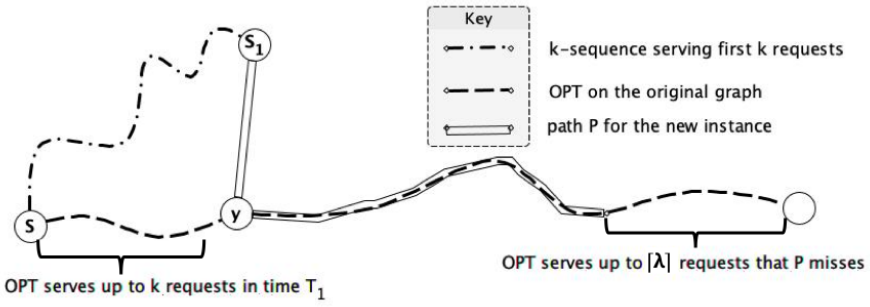

Fig. 1. An illustration of the paths taken by OPT and k-SEQ in Theorem 4. T_1 is the time needed for k-SEQ to serve its initial group of k requests, ending at s_1. The first node on the path of OPT after time T_1 is y. A feasible path P starting at time T_1 is from s_1 to y and then proceeds to the right. (It is possible for s_1 and y to be collocated.)

[§]Note that when the graph is complete, t_{max} (t_{min}) is the maximum (minimum) distance over all pairs of nodes. Otherwise, using the pre-processing described in the footnote [†] in Section 2, we have that the distance between any two non-adjacent nodes is the shortest distance between those nodes, and t_{max} (t_{min}) is the maximum (minimum) distance over all of these distances.

3.1 k-Sequence upper bound for large λ

We will now show that for sufficiently large constant λ, the k-SEQ algorithm is a $(1+\lambda/k)$-approximation, a better ratio than the result obtained in Theorem 4 for sufficiently large λ. However, Theorem 4 remains better when $2+\lambda/k < k$. We note that when $1+\lambda/k \geq k$, this upper bound of $(1+\lambda/k)$ matches the lower bound which will be discussed in Section 4.

Theorem 5. *For any instance I of TDARP,*

$$|\text{OPT}(I)| \leq \max\{(1+\lambda/k)|k\text{-SEQ}(I)| + \lambda(k-1)/k + 1, k|k\text{-SEQ}(I)| + k\}.$$

I.e., when $1+\lambda/k \geq k$, we have $|\text{OPT}(I)| \leq (1+\lambda/k)|k\text{-SEQ}(I)| + \max\{\lambda(k-1)/k+1, k\}$, so k-SEQ is a $(1+\lambda/k)$-approximation in this case.[¶]

Proof. Let $m = |\text{OPT}|$, and $n = |k\text{-SEQ}|$. Suppose that $m < kn + k$. Then $|\text{OPT}| < kn+k = k|k\text{-SEQ}|+k$, giving our desired result. Thus, for the remainder of the proof, we assume that $m \geq kn + k$, and proceed to show that $|\text{OPT}| < (1+\lambda/k)|k\text{-SEQ}| + \lambda(k-1)/k + 1$.

Let the OPT path serve, in order, requests $r_1, r_2, \ldots, r_m$, whose respective service times are $y_1, y_2, \ldots, y_m$. Let x_j be the time taken by an empty drive required between request r_{j-1} and request r_j for $2 \leq j \leq m$, and let x_1 be the time taken to get from the origin to request r_1. Note that any x_j may be 0. Thus, the driveTime taken by the OPT path is:

$$x_1 + y_1 + x_2 + y_2 + \cdots + x_m + y_m \leq T. \tag{2}$$

Let $r_{m+1}, r_{m+2}, \ldots$ be some fixed arbitrary labeling of the requests not served by OPT. Now we consider the k-SEQ algorithm and denote the requests served by k-SEQ as $r_{\alpha_1}, \ldots, r_{\alpha_n}$. Denote by q the number of times that k-SEQ searches for the fastest sequence of k requests to serve; $q = \lceil n/k \rceil$. Then $n = k(q-1)+\rho$ for some $1 \leq \rho \leq k$.

By Lemma 2 in the Appendix, we can find $q-1$ disjoint subsequences of k consecutive integers from $\{1, \ldots, m\}$, and a qth disjoint subsequence of ρ consecutive integers from $\{1, \ldots, m\}$, i.e. for $i_1 = 1$ we have:

$$i_1, \ldots, i_1+k-1, \quad \ldots, \quad i_{q-1}, \ldots, i_{q-1}+k-1, \quad i_q, \ldots, i_q+\rho-1 \quad \text{where}$$

$$\{i_j, ..., i_j+k-1\} \cap \{\alpha_1, \ldots, \alpha_{k(j-1)}\} = \emptyset, \text{ for } 2 \leq j \leq q-1,$$
$$\{i_j, ..., i_j+\rho-1\} \cap \{\alpha_1, \ldots, \alpha_{k(j-1)}\} = \emptyset, \text{ for } j = q.$$

Since both k-SEQ and OPT start at the same origin, the greedy nature of k-SEQ ensures the time k-SEQ spends on its first set of k requests, including any empty drives to those requests, is at most $x_1 + y_1 + \cdots + x_k + y_k$. By Lemma 2 we know $\{i_j, \ldots, i_j+k-1\}$ is disjoint from $\{\alpha_1, \ldots, \alpha_{k(j-1)}\}$, so for the jth set of k requests with $2 \leq j \leq q-1$, the path resulting from going to requests

[¶]Note that if $1+\lambda/k \geq k$, then $\lambda(k-1)/k+1 \geq k$.

$\{r_{i_j}, \ldots, r_{i_j+k-1}\}$ is available and so by the greedy nature of k-SEQ, the time spent by k-SEQ is at most $t_{max} + y_{i_j} + x_{i_j+1} + \cdots + y_{i_j+k-1}$, since t_{max} is the maximum time needed to get to request r_{i_j}. And finally by the same reasoning the time spent by k-SEQ on the last set of ρ requests, still including any drives to those requests, is at most $t_{max} + y_{i_q} + x_{i_q+1} + \cdots + y_{i_q+\rho-1}$. Thus, the total time spent by k-SEQ is at most

$$\begin{aligned} T_0 :=& (q-1)t_{max} + x_1 + y_1 + \cdots + x_k + y_k \\ &+ \sum_{j=2}^{q-1} (y_{i_j} + x_{i_j+1} + \cdots + y_{i_j+k-1}) + y_{i_q} + x_{i_q+1} + \cdots + y_{i_q+\rho-1}. \end{aligned} \tag{3}$$

Now, let r_J be any request served by OPT where J is not any of the indices appearing in the right hand side of (3). If $T_0 + t_{max} + y_J \leq T$, then k-SEQ could have served another request, a contradiction. Therefore, we must have $T_0 + t_{max} + y_J > T$. Combining this observation with (2), we have:

$$\begin{aligned} &(q-1)t_{max} + x_1 + y_1 + \cdots + x_k + y_k + \sum_{j=2}^{q-1} (y_{i_j} + x_{i_j+1} + \cdots + y_{i_j+k-1}) + y_{i_q} \\ &+ x_{i_q+1} + \cdots + y_{i_q+\rho-1} + t_{max} + y_J > x_1 + y_1 + \cdots + x_m + y_m. \end{aligned} \tag{4}$$

By construction, in the left hand side of (4), the x terms all have distinct indices, the y terms all have distinct indices, and these terms also appear on the right hand side.

Let $\mathcal{I}$ be the set of these indices on the left hand side. So $\mathcal{I} \subseteq \{1, \ldots, m\}$. Then subtracting these terms from both sides of the equation yields $qt_{max} > x_J + \sum\{x_j : j \in \{1, \ldots, m\} \backslash \mathcal{I}\} + \sum\{y_j : j \in \{1, \ldots, m\} \backslash \mathcal{I}\}$, so: $qt_{max} > \sum\{y_j : j \in \{1, \ldots, m\} \backslash \mathcal{I}\}$. Since $|\mathcal{I}| = n + 1$, there are $m - n - 1$ of the y_j terms on the right hand side. Since each $y_j \geq t_{min}$, we have $qt_{max} > (m - n - 1)t_{min}$. Thus $q\lambda > m - n - 1$. Because $q = \lceil n/k \rceil$, then $q \leq (n + k - 1)/k$. Then $m \leq (n + k - 1)\lambda/k + n + 1 \leq (1 + \lambda/k)n + \lambda(k - 1)/k + 1$ as desired. ■

Note that for $k = 1$, k-SEQ is the polynomial time algorithm that repeatedly finds and serves the quickest request. Theorem 5 (in this section) and Theorem 6 (in the next section) yield the following corollary regarding this algorithm.

Corollary 1. 1-SEQ *(i.e., k-SEQ with $k = 1$) has approximation ratio $1 + \lambda$, which is tight for all λ (see Figure 2 for an illustration of the lower bound).*

4 k-Sequence lower bound

We now present lower bounds on k-SEQ; specifically, the lower bound is $1 + \lambda$ for $k = 1$, shrinking to 2 for $k = \lambda$, and shrinking further towards 9/7 for $k > \lambda$. Note that Theorem 6 matches the upper bound of Theorem 5 when $1 + \lambda/k \geq k$.

Theorem 6. *The approximation ratio of k-SEQ for TDARP has lower bound $1 + \lambda/k$.*

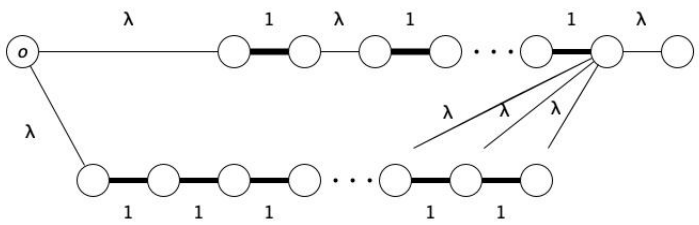

Fig. 2. The instance described in Theorem 6 when $k = 1$. Note that the graph is complete, and any edge (u, v) that is not shown has distance equal to the minimum of λ and the shortest-path distance along the edges shown between u and v. The bold edges represent requests. k-SEQ serves requests along the top path while OPT serves along the bottom.

Proof. Consider an instance (see Figure 2 for the case of $k = 1$) where there are two "paths" of interest, both a distance of λ away from the origin. Any edge (u, v) that is not shown has distance equal to the minimum of λ and the shortest-path distance along the edges shown between u and v. There is one long chain of T requests, which is the path chosen by the optimal solution (the bottom path in Fig. 2), and another "broken" chain (the top path in Fig. 2) that consists of k sequential requests at a time with a distance of λ from the end of each chain to any other request in the instance. (Generalizing Figure 2, for $k > 1$, these requests occur in chains of length k instead of single requests.) Note this graph satisfies the properties of a metric space.

The algorithm may choose to follow the path of the broken chain, serving k requests at a time, but being forced to move a distance of λ between each k-chain. In this manner, for every k requests served, the algorithm requires $k+\lambda$ units of time, while the optimal solution can serve k requests every k time units (after the first λ time units). Thus the approximation ratio of k-SEQ is at least $1 + \lambda/k$. ■

We now show, however, that as k grows relative to λ the ratio of k-SEQ improves but does not reach (or go below) 9/7.

Theorem 7. *The approximation ratio of k-SEQ has lower bound no better than 9/7 for any $k > \lambda$.*

Proof. Refer to Figure 3 with nodes $o, a_0, a_1, \ldots, a_L$, $b_{k+\lambda+1}, \ldots, b_L$. The distances are: o is λ away from every node, $dist(a_{i-1}, a_i) = 1$ for all $i = 1, \ldots, L$, and $dist(a_i, b_i) = dist(b_i, a_i) = 1$ for all $i = K + \lambda + 1, ..., L$.

Consider the edges shown in Figure 3 (Top) as forming a connected spanning subgraph G'. Define the distance between any two nodes whose distance is not yet defined as the length of the shortest path within G' between the two nodes, capping the distance at λ; that is, for any $i \neq j$, $dist(a_i, a_j) = \min\{\lambda, |i - j|\}$ and $dist(a_i, b_j) = \min\{\lambda, |i - j| + 1\}$. A distance defined this way satisfies the properties of a metric space.

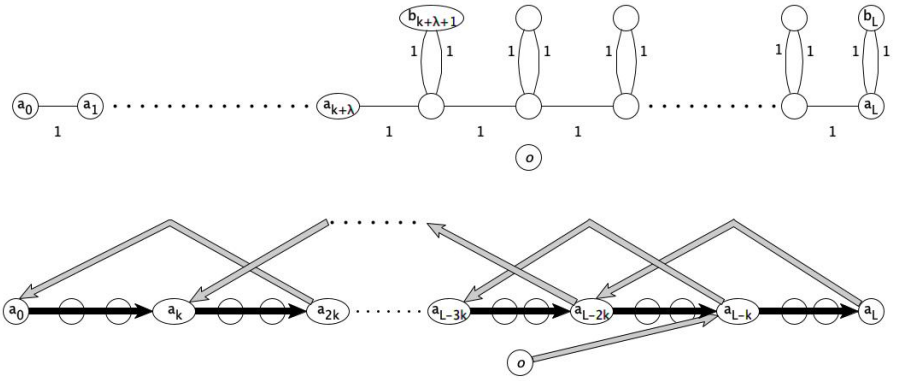

Fig. 3. Top: An instance where OPT serves no fewer than 9/7 the number of requests served by k-SEQ. Bottom: A depiction of the top instance illustrating the path taken by k-SEQ. Bold edges indicate requests served. k-SEQ starts at o, serves the k requests from a_{L-k} to a_L, and then spends time moving to a_{L-2k} to serve the next collection of k requests, continuing similarly until the time limit. In both figures the graph is complete but only relevant edges are shown.

The requests are: (a_{i-1}, i) for $i = 1, \ldots, L$ (the "spine"), (a_i, b_i) and (b_i, a_i) for $i = k+\lambda+1, k+\lambda+2, \ldots, L$ (a "loop"). Note there are no loops for $i \leq k+\lambda$.

OPT serves all requests via the path $o, a_0, a_1, a_2, \ldots, a_{k+\lambda}, a_{k+\lambda+1}, b_{k+\lambda+1}, a_{k+\lambda+1}, a_{k+\lambda+2}, b_{k+\lambda+2}, ..., a_L$. This serves $L + 2(L - k - \lambda) = 3L - 2\lambda - 2k$ requests in time $\lambda + 3L - 2\lambda - 2k = 3L - \lambda - 2k$. Meanwhile, k-SEQ will serve the path that begins with the segment $o, a_{L-k}, a_{L-k+1}, \ldots, a_L$, followed by an empty drive to the segment $a_{L-2k}, a_{L-2k+1}, \ldots, a_{L-k}$, followed by an empty drive, and so on, until the final segment of k requests $a_0, a_1, \ldots, a_k$.

Note that because $2k > \lambda$, then $dist(a_L, a_{L-2k}) = \lambda$. So the entire k-SEQ path then takes total time $(\lambda + k)L/k$ since there are L/k segments, and k-SEQ initially serves L requests during these $(\lambda + k)L/k$ units of time. There is time remaining, namely $T' = 3L - \lambda - 2k - (\lambda + k)L/k = (3 - (\lambda + k)/k)L - \lambda - 2k$. Since $k > \lambda/2$, we have $(\lambda + k)/k < 3$, so T' is positive for large enough L. There are now disconnected two-cycles $(a_i, b_i), (b_i, a_i)$ for $i = \lambda + k + 1, ..., L$ left for k-SEQ. With time T' left, k-SEQ is now at a_k. Note that these are all distance λ away from a_k. There are two cases based on the parity of k.

Case 1: k is even. Moving to the group (i.e. a sequence of k requests consisting of $k/2$ consecutive two-cycles) $\{(a_i, b_i), (b_i, a_i)$ for $i = j, \ldots, j + k/2 - 1\}$ for any j with $k + \lambda + 1 \leq j \leq L - k/2 + 1$ serves k requests in time $k + \lambda + k/2 - 1 = \lambda + 3k/2 - 1$ because of the required empty drive of time λ to get to the first request and the $(k/2 - 1)$ empty drives between the $k/2$ two-cycles. Then k-SEQ from a_k would move to this group with $j = L - k/2 + 1$, followed by this group with $j = L - k + 1$, and so on, subtracting $k/2$ from j each time. Thus k-SEQ serves $T'k/(\lambda + 3k/2 - 1)$ additional requests in the remaining time T'; note for simplicity we can choose L so that $T'k$ is evenly divisible by $(\lambda + 3k/2 - 1)$.

Case 2: k is odd. The behavior of k-SEQ is similar to the even k case except that in each iteration, k-SEQ serves $(k-1)/2$ two-cycles and one additional request. Specifically, from a_k, k-SEQ would move to the group of k requests

$$\{(a_j, b_j),(b_j, a_j), (a_{j+1}, b_{j+1}), (b_{j+1}, a_{j+1}), \ldots, (a_{j+(k-1)/2-1}, b_{j+(k-1)/2-1}),$$
$$(b_{j+(k-1)/2-1}, a_{j+(k-1)/2-1}), (a_{j+(k-1)/2}, b_{j+(k-1)/2})\}$$

where we here set $j = L-(k-1)/2$. This serves k requests in time $\lambda + k + (k-1)/2 = \lambda + 3k/2 - 1/2$. But the next group of k requests would be (still setting $j = L-(k-1)/2$, and continuing from $b_{j+(k-1)/2}$):

$$\{(b_{j-(k-1)/2}, a_{j-(k-1)/2}),(a_{j-(k-1)/2+1}, b_{j-(k-1)/2+1}), (b_{j-(k-1)/2+1}, a_{j-(k-1)/2+1}),$$
$$(a_{j-(k-1)/2+2}, b_{j-(k-1)/2+2}), \ldots, (b_{j-1}, a_{j-1})\}.$$

This group serves k requests in time $k + \lambda + (k-1)/2 - 1 = \lambda + 3k/2 - 3/2$. Together these two sequences serves $2k$ requests in time $2\lambda + 3k - 2$. Then k-SEQ repeats these two sequences with j decreasing by k each time until time runs out. Thus k-SEQ serves $T'2k/(2\lambda + 3k - 2)$ additional requests in the remaining time T'. Note this is identical to the case of even k.

In both cases, k-SEQ serves a total of $\frac{L+T'\cdot k}{(\lambda+3k/2-1)} = \frac{(7kL-2L-2k\lambda-4k^2)}{(3k+2\lambda-2)}$ requests. Then $\frac{|\text{OPT}|}{|k\text{-SEQ}|}$ is $\frac{(3L-2\lambda-2k)(3k+2\lambda-2)}{7kL-2L-2k\lambda-4k^2}$. As L grows, this approaches $\frac{3(3k+2\lambda-2)}{(7k-2)}$. Note that because $\lambda \geq 1$ and $k \geq 1$, this ratio is $\geq 9/7$; thus $9/7$ is a lower bound. ■

Note that when $k \leq \lambda$, Theorem 6 gives a lower bound of $1 + \lambda/k \geq 2$; so we have a lower bound of $9/7$ for any k, λ.

5 Final remarks

Observe that if we let N denote the maximum number of requests that can be served within time t_{max}, then it is possible to show that our upper bound theorems above hold with λ replaced by $N+1$ and the lower bound theorems hold with λ replaced by N. Note that $N \leq \lambda$; the hypothetically modified upper bound theorems would be improvements in the case where $N + 1 < \lambda$. Additionally, we could have defined t_{min} as the minimum request service time when there is at least one request, leaving t_{max} as the maximum edge weight, and the theorems above would still hold.

It remains open whether our lower bound of $9/7$ from Theorem 7 is tight when $k > \lambda$. Another open problem is to close the gap between the upper and lower bounds in the approximation ratio when $k \leq \lambda \leq k(k-1)$.

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:

Cacchiani, V., Marchetti-Spaccamela, A. (eds.) 19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, (ATMOS 2019). vol. 75, pp. 11:1–11:15 (2019)
2. Archer, A., Bateni, M., Hajiaghayi, M., Karloff, H.: Improved approximation algorithms for prize-collecting Steiner tree and TSP. SIAM Journal on Computing **40**(2), 309–332 (2011)
3. Balas, E.: The prize collecting traveling salesman problem. Networks **19**(6), 621–636 (1989)
4. Bienstock, D., Goemans, M.X., Simchi-Levi, D., Williamson, D.: A note on the prize collecting traveling salesman problem. Mathematical programming **59**(1-3), 413–420 (1993)
5. Blum, A., Chawla, S., Karger, D.R., Lane, T., Meyerson, A., Minkoff, M.: Approximation algorithms for orienteering and discounted-reward TSP. SIAM Journal on Computing **37**(2), 653–670 (2007)
6. Charikar, M., Motwani, R., Raghavan, P., Silverstein, C.: Constrained TSP and low-power computing. In: Workshop on Algorithms and Data Structures. pp. 104–115. Springer (1997)
7. Christman, A., Chung, C., Jaczko, N., Milan, M., Vasilchenko, A., Westvold, S.: Revenue Maximization in Online Dial-A-Ride. In: 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017). vol. 59, pp. 1:1–1:15. Dagstuhl, Germany (2017)
8. City of Plymouth, Minnesota: Plymouth metrolink dial-a-ride. Personal communication, http://www.plymouthmn.gov/departments/administrative-services-/transit/plymouth-metrolink-dial-a-ride
9. Cordeau, J.F., Laporte, G.: The dial-a-ride problem: models and algorithms. Annals of Operations Research **153**(1), 29–46 (Sep 2007)
10. Elbassioni, K., Fishkin, A.V., Mustafa, N.H., Sitters, R.: Approximation algorithms for euclidean group TSP. In: International Colloquium on Automata, Languages, and Programming. pp. 1115–1126. Springer (2005)
11. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM Journal on Computing **24**(2), 296–317 (1995)
12. Metropolitan Council: Transit link: Dial-a-ride small bus service. Personal communication, https://metrocouncil.org/Transportation/Services/Transit-Link.aspx
13. Molenbruch, Y., Braekers, K., Caris, A.: Typology and literature review for dial-a-ride problems. Annals of Operations Research **259**(1), 295–325 (2017)
14. Paul, A., Freund, D., Ferber, A., Shmoys, D.B., Williamson, D.P.: Prize-collecting TSP with a budget constraint. In: 25th Annual European Symposium on Algorithms (ESA 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
15. Paul, A., Freund, D., Ferber, A., Shmoys, D.B., Williamson, D.P.: Budgeted prize-collecting traveling salesman and minimum spanning tree problems. Mathematics of Operations Research **45**(2), 576–590 (2020)
16. Stagecoach Corporation: Dial-a-ride. Personal communication, http://stagecoach-rides.org/dial-a-ride/

6 Appendix

6.1 Lower Bound Instance from Proof in Section 2.2

The following theorem directly adapts the lower bound instance in [7] for online SBP to our work in the offline setting. The intuition behind the bound carrying

over lies in the observation that in the online instance no requests were released after time 0.

Theorem 8. SBP *serves no more than* OPT/4 *requests.*

Proof. Consider an instance (S, T, o). For some even f, there are T requests and each request requires $(T/(2f)) + 1$ time to serve. OPT serves $\frac{T}{(T/(2f))+1}$ requests. SBP serves during every odd time segment, and due to the time required for each request, can serve only one request per time segment. So in total SBP can serve at most $f/2$ requests. We have:

$$\frac{\text{OPT}(S,T,o)}{\text{SBP}(S,T,o)} \geq \frac{\left(\frac{T}{(T/(2f))+1}\right)}{f/2} \geq \frac{4T}{T+2f} \tag{5}$$

As $T \to \infty$, the above expression approaches 4.

6.2 Proof and Figure for Section 2.1.1

We now show that no polynomial-time algorithm can be guaranteed to serve the optimal number of requests, even when the time limit for the algorithm is augmented by any constant factor $c \geq 1$.

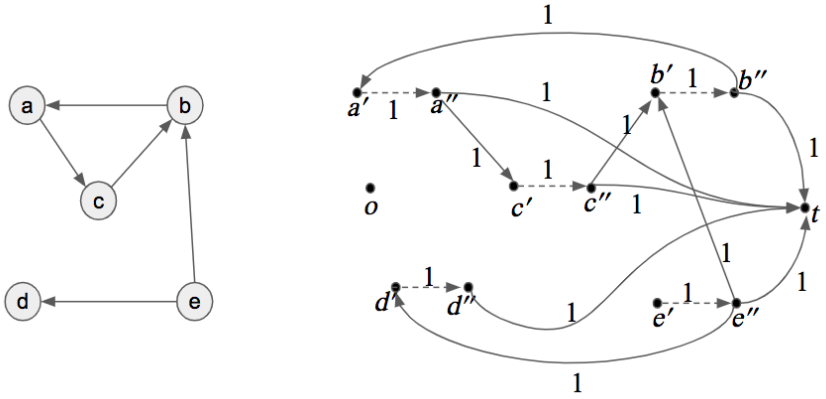

Fig. 4. An example instance G of the directed Hamiltonian path problem where $n = 5$ (left), and the graph G' of the corresponding instance for TDARP where $T = 2n + 1$ (right). Four types of edges have distance 1: (1) all edges (v', v''), (2) for any $(u, v) \in G$, the edge (u'', v'), (3) for all nodes v' in G', the edge (o, v'), and (4) for all nodes v'' in G', the edge (v'', t). All other edges to make the graph complete (not shown) have distance $c(2n + 1) + 1$ for some $c \geq 1$.

Proof. We will show that a polynomial-time *c-time-approximation* to TDARP yields a polynomial-time decider for the directed Hamiltonian path problem. Portions of this proof are inspired by the NP-hardness proof in [1] where the problem of TDARP was studied on the uniform metric space.

Given a directed Hamiltonian path problem input $G = (V, E)$ where $n = |V|$, we build an instance I for TDARP as follows. First, construct a complete graph G' with $2n + 2$ nodes (see Figure 4; while G' is a complete graph, we show only the edges of interest, omitting those with distance $c(2n + 1) + 1$): one node will be the server origin o, one will be a designated "sink" node t, and the other $2n$ nodes are as follows. For each node $v \in V$, create a node v' and a node v'' in G'. Four types of edges have distance 1 in G': (1) all edges (v', v''), (2) for any $(u, v) \in G$, the edge (u'', v'), (3) for all nodes v' in G', the edge (o, v'), and (4) for all nodes v'' in G' the edge (v'', t) . All other edges have distance $c(2n+1)+1$ for some $c \geq 1$. To construct the set of requests S, create a TDARP request in G' from node v' to node v'' for each $v \in V$, which we will refer to as a *node-request.* Further, for each edge $(u, v) \in E$, create a TDARP request from node u'' to node v' in G', which we will refer to as an *edge-request.* Additionally, for each $v \in V$, create an edge-request from v'' to the designated sink node t in G'. Let $I = (G', S, T = 2n + 1)$.

By way of contradiction, we assume there is a *c-time-approximation* for TDARP called ALG and let $I' = (G', S, cT = c(2n + 1))$ be an instance of TDARP where G' and S are as described above. We claim ALG can be used as a polynomial-time decider for the Hamiltonian path problem; specifically, G has a Hamiltonian path if and only if $\text{ALG}(I') \geq 2n$

Suppose $\text{ALG}(I') \geq 2n$. Consider a sequence of $2n$ requests in G' that takes time at most $c(2n+1)$. Note that any such sequence may not contain any of the edges with distance $c(2n + 1) + 1$. Further note that by construction of G', any such sequence of TDARP requests must alternate between node-requests and edge-requests, where any edge to the sink is counted as an edge-request (and must be a terminal request). Since destinations in G' can be partitioned into the sink, single-primed nodes, and double-primed nodes, we can thus analyze the three possibilities for the destination of the final TDARP request.

If either the sink or a single-primed node is the destination for the final TDARP request, the TDARP sequence must end with an edge-request. The alternating structure ensures the TDARP sequence begins with a node-request, and contains exactly n node-requests and n edge-requests. If a double-primed node is the destination for the final TDARP request, the TDARP sequence must end with a node request. The alternating structure ensures the TDARP sequence begins with an edge-request, and contains exactly n edge-requests and exactly n node requests. Thus, the TDARP sequence always contains n node requests. This ensures that the length n path in the original graph G includes all n nodes in the original graph G, and thus the existence of a Hamiltonian path. This shows $\text{ALG}(I') \geq 2n$ implies the existence of a Hamiltonian path in G.

For the other direction of the proof, we show that if there is a Hamiltonian path in G then $\text{ALG}(I') \geq 2n$. Let $p = (v_1, v_2, \ldots, v_n)$ be a Hamiltonian path

in G. Construct the sequence of $2n$ TDARP requests in G' by the node request from v'_1 to v''_1, the edge-request from v''_1 to v'_2, the node request from v'_2 to v''_2, the edge-request from v''_2 to v'_3, and so forth, through the edge-request from v''_{n-1} to v'_n, the node request from v'_n to v''_n, and finally the edge-request from v''_n to the designated sink t. The entire sequence takes time $2n+1$, so $\text{OPT}(I) = 2n$. By definition of a *c-time-approximation*, this implies that $\text{ALG}(I') \geq 2n$. ■

6.3 Definitions and Proof for Section 2.1.2

Let G_1 and G_2 be complete subgraphs of a fixed complete graph with requests. We say G_1 and G_1 are *request-isomorphic* if there is a correspondence of nodes that preserves edge weights and request status.

We define an *inductive stateless greedy* algorithm on an offline DARP problem to have the following properties.

1. Inductive. The algorithm may make calculations and decisions only when it (the server) is at a node. A decision is a *chosen path* emanating from the node where the server is; the algorithm must complete the first drive on the chosen path, but may choose a different chosen path at any subsequent node on the path. Note that at the start of the algorithm and whenever the algorithm reaches the end of a chosen path, the algorithm must make a new decision.
2. Stateless. At each node the algorithm visits, the algorithm does not remember anything from previous calculations other than the current chosen path.
3. Greedy. The algorithm makes decisions based on comparing possible paths emanating from its current location. That is, the algorithm computes an objective function based on the nodes of each candidate path and any nodes from the current chosen path, but no other nodes. The algorithm chooses a path P_0 that optimizes this objective function. Note that if two paths Q and Q' have the property that $Q \cup P_0$ and $Q' \cup P_0$ are request-isomorphic via an isomorphism that fixes the chosen path P_0, then the objective function cannot distinguish between Q and Q'.

Theorem 2. *Let M be a constant and let* ALG *be a deterministic inductive stateless greedy algorithm such that the algorithm considers only candidate paths with at most M edges. If λ is not bounded, then* ALG *has an unbounded approximation ratio.*

Proof. Consider the following instance. Without loss of generality let M be an integer (replacing M by $\lceil M \rceil$ if needed). We will construct an instance with integer edge weights where the minimum edge weight will be 1 and hence maximum edge weight will be λ. Let $T = \lambda + n$, where n is a positive integer. The instance depends on integers M, n, λ.

Consider all possible request-isomorphism classes of nonempty complete graph instances with M nodes or fewer, for which edges can have only integer value weights from 1 to λ and there are M or fewer requests. Clearly there are only a

finite number of such isomorphism classes. Create a collection Φ of disjoint complete graphs such that there are $2M + T + 1$ of each isomorphism class. Define the distance between any pair of nodes from two different elements of Φ to be λ.

The optimal schedule OPT will serve P, a chain of n requests of distance 1 each, disjoint from all nodes in Φ. Define the distance from any node in P to any node in Φ to be λ.

Let o be the origin, a node that is distance λ from every node in P or in Φ. This completes the definition of our instance. (It is a metric space if we restrict Φ to contain only metric spaces.)

We claim that ALG cannot be guaranteed to ever visit P. We prove the claim by induction on how many nodes ALG has visited so far. Since ALG starts at o, the base case holds. For the induction step, suppose ALG is at a node b not on P and ALG has not visited any nodes of P so far and that its current chosen path does not include any nodes in P. If ALG does not make a new decision here, then ALG will continue to not have visited P when it drives to the next node and we are done. So suppose ALG makes a new decision here.

Let P_0 be the current chosen path that includes b (so if we are at the start of the algorithm, just let $P_0 = \{o\}$); ALG (at b) could be at the last node of P_0. Let Q be a path of at most M edges emanating from b that optimizes ALG's objective function. We now show that if Q has nodes that are in P, then there exists a path Q' emanating from b that has the same objective function value but Q' has no nodes in P. Let Q be the path $A_1, Q_1, A_2, Q_2, ..., Q_r, A_{r+1}$ where A_i's are sequence of nodes not in P and Q_i's are sequence of nodes in P. It is possible that nodes may be repeated, and it is possible that some A_i's and Q_i's consist of single nodes; they are nonempty except for possibly A_{r+1}. Note that the distance going from an A_i to a Q_i and from a Q_i to an A_{i+1} is λ.

Consider the complete subgraph spanned by the set of nodes $\hat{Q} = Q_1 \cup \cdots \cup Q_r$. Because Q has at most $M+1$ nodes and one of them is b, then there are M or fewer nodes in $\hat{Q}$. Also, $\hat{Q}$ has at most $M - 1 \leq M$ requests. Then the complete subgraph spanned by $\hat{Q}$ is request-isomorphic to $2M + T + 1$ of the graphs in the collection Φ. Because ALG could have visited at most $T - 1$ of these in time T and P_0 has at most $M + 1$ nodes and there are at most M nodes in the union of A_i's, then there is at least one element $\hat{Q}'$ from Φ that is request-isomorphic to $\hat{Q}$ with the property that $\hat{Q}'$ has no nodes visited by ALG so far and no nodes in P_0 or the A_i's. Note that every node in $\hat{Q}'$ is distance λ from every node in P_0 and the A_i's and that the requests in $\hat{Q}'$ are not yet served. Then each Q_i has a corresponding $Q'_i \subseteq \hat{Q}'$.

Consider the path $Q' = A_1, Q'_1, A_2.Q'_2, ..., Q'_r, A_{r+1}$. Because the distance from every node in Q_i or Q'_i to every node in A_j and P_0 is λ, then $Q \cup P_0$ is request-isomorphic to $Q' \cup P_0$. Thus the objective function must evaluate to the same value on Q' as Q. Thus Q' also optimizes the objective function. Then ALG cannot guarantee that it does not choose Q' as its chosen path. This finishes the proof of the claim that ALG cannot be guaranteed to ever touch P.

Since ALG is never on P, then ALG must alternate between empty drives of distance λ and sequences of drives within each graph from the collection Φ.

Since each graph from Φ has at most M requests, then ALG alternates between an empty drive and a sequence of at most M requests. Let E denote the number of such empty drives. Since there are no requests at the origin, and ALG can serve at most M requests before being forced to go to another element of Φ, we have $|ALG| \leq EM$. Thus, $T = E\lambda + |\text{ALG}| \geq (|\text{ALG}|/M)\lambda + |\text{ALG}|$, which implies ALG can serve at most $TM/(\lambda + M) = (n + \lambda)M/(\lambda + M)$ requests.

Since OPT serves n requests, then

$$\frac{|\text{OPT}|}{|\text{ALG}|} \geq \frac{n(\lambda + M)}{(n + \lambda)M}.$$

Note the limit as $n \to \infty$ of the expression on the right hand side is $(\lambda + M)/M$. By taking n sufficiently large, we can say

$$\frac{|\text{OPT}|}{|\text{ALG}|} \geq \frac{\lambda + M}{M} - 1.$$

This ratio is unbounded as λ increases.

6.4 Proofs for Theorem 5 in Section 5

The following lemmas are used for the proof of Theorem 5.

Lemma 1. *Suppose we have d disjoint ordered sequences, some which may be empty, consisting of b elements in totality and are given c elements, some of which may not occur in the d sequences. If for some h, $b \geq ch + dh - d + 1$, then there exists a subsequence of h consecutive elements in one of the sequences which does not include the indicated c elements.*

Proof. Note that if one or more of the c elements is not in any of the d ordered sequences, then we would just ignore them and the resulting set of elements to avoid would be smaller so the inequality in the lemma would still be satisfied. So for simplicity, in this proof we assume the c elements to be avoided are inside the set of b elements.

First we prove the case of $d = 1$. The c elements partition the one given sequence into $(c + 1)$ subsequences (some possibly empty). If, by way of contradiction, each of these subsequences has length $h - 1$ or less, then the total number of elements is at most $(c + 1)(h - 1) + c = ch + h - 1$, a contradiction since we are given $b \geq ch + h$, completing the proof for the case of $d = 1$.

Now we prove the general case. Let the d subsequences be $B_1, \ldots, B_d$ and suppose the c elements are distributed as sets $C_1, \ldots, C_d$ inside these. Suppose by way of contradiction that $|B_j| \leq |C_j|h + h - 1$ for each j. Then summing these inequalities yields

$$b = \sum_{j=1}^{d} |B_j| \leq \sum_{j=1}^{d} (|C_j|h + h - 1) = ch + d(h - 1) < ch + dh - d + 1,$$

a contradiction. Hence $|B_j| \geq |C_j|h + h$ for some j and the first case of $d = 1$ implies this B_j contains the desired subsequence of h consecutive elements. ■

Lemma 2. *Let $m \geq nk+k$, $q = \lceil n/k \rceil$, and $n = k(q-1)+\rho$. Note this implies $1 \leq \rho \leq k$. Suppose we have a sequence $\alpha_1, \ldots, \alpha_n$ of distinct integers. We can find $q-1$ disjoint subsequences of k consecutive integers from $\{1, \ldots, m\}$, and a qth disjoint subsequence of ρ consecutive integers from $\{1, \ldots, m\}$, i.e. we have:*

$$i_1, \ldots, i_1 + k - 1, \quad \ldots, \quad i_{q-1}, \ldots, i_{q-1} + k - 1, \quad i_q, \ldots, i_q + \rho - 1 \tag{6}$$

such that $i_1 = 1$ and we have

$$\{i_j, ..., i_j + k - 1\} \cap \{\alpha_1, \ldots, \alpha_{k(j-1)}\} = \emptyset, \text{ for } 2 \leq j \leq q-1, \tag{7}$$

$$\{i_j, ..., i_j + \rho - 1\} \cap \{\alpha_1, \ldots, \alpha_{k(j-1)}\} = \emptyset, \text{ for } j = q. \tag{8}$$

Proof. Set $i_1 = 1$. We now show how we can choose i_q, then i_{q-1}, and so forth, down to i_2. We need to enforce that the $i_1, \ldots, i_n \in \{1, \ldots, m\}$ are distinct and that the conclusion ((7) and (8)) of the lemma are true. We start by choosing i_q so that the length-ρ sequence $\{i_q, \ldots, i_q + \rho - 1\}$ avoids $\{\alpha_1, \ldots, \alpha_{k(q-1)}\}$ and $\{i_1, \ldots, i_k\} = \{1, \ldots, k\}$. That is, we want to choose ρ consecutive integers from $\{k+1, \ldots, m\}$ (which has size $m-k$) while avoiding those in $\{\alpha_1, \ldots, \alpha_{k(q-1)}\}$ (which has size $k(q-1)$). We apply Lemma 1 with $d = 1$ and $b = m-k$ and $c = (q-1)k = n-\rho$. Since $m \geq kn+k$, we have that $m-k \geq kn \geq kn-\rho(\rho-1) \geq \rho n - \rho^2 + \rho = \rho(n-\rho)+\rho$, ensuring that $b \geq \rho c + \rho$, and thus that there is a set of ρ consecutive integers in $\{k+1, \ldots, m\}$ such that (7) is satisfied.

Having chosen an acceptable i_q, we now proceed to choose an i_{q-1} such that

$$\{i_{q-1}, \ldots, i_{q-1} + k - 1\} \subseteq \{k+1, \ldots, m\} \backslash \{i_q, \ldots, i_q + \rho - 1\}$$

avoids $\{\alpha_1, \ldots, \alpha_{k(q-2)}\}$. Note that since $\{i_q, \ldots, i_q + \rho - 1\}$ is a sequence of consecutive integers, $\{k+1, \ldots, m\} \backslash \{i_q, \ldots, i_q + \rho - 1\}$ consists of two sequences (one possibly empty) of consecutive numbers. Thus we can apply Lemma 1 with $d = 2$, $b = m-k-\rho$, $c = k(q-2)$ and $h = k$. We calculate that $ch+dh-d+1 = k^2(q-2)+2k-2+1 = k^2q-2k^2+2k-1 = k^2q-(k-1)^2-k^2$ and $b = m-k-\rho \geq nk+k-k-\rho = (k(q-1)+\rho)k-\rho = k^2q-k^2+\rho(k-1)$. Since $b \geq k^2q-k^2+\rho(k-1)$ and $\rho(k-1) \geq 0 \geq -(k-1)^2$, we have $b \geq k^2q-(k-1)^2-k^2 = ch+dh-d+1$ and so by Lemma 1, there exists an i_{q-1} that we desire.

We similarly obtain $i_{q-2}, \ldots, i_2$ and the corresponding collections of consecutive requests. In each step, d increases by 1, b decreases by k and c decreases by k, so that that for finding i_{q-j} (where $1 \leq j \leq q-2$), we have $d = j+1$, $b = m-\rho-jk$, $c = k(q-j-1)$. We just need to verify the hypothesis of Lemma 1 to finish this proof. We have $ch+dh-d+1 = k^2(q-j-1)+(j+1)k-(j+1)+1 = k^2q+j(-k^2+k-1)-k^2+k = k^2q-j(k-1)^2-jk-k^2+k$ whereas $b = m-\rho-jk \geq nk+k-\rho-jk = (k(q-1)+\rho)k-\rho-jk+k = k^2q-k^2+\rho(k-1)-jk+k$. It is now clear that $b \geq ch+dh-d+1$, and the proof is complete. ∎