Serving rides of equal importance for budgeted 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 revenue earned for serving it. The goal for the server is to serve requests within a given time limit so as to maximize the total revenue. We consider the uniform-revenue variant (equivalent to maximizing the number of requests served), whose applications include paratransit services. We first show that no polynomial-time algorithm can be guaranteed to earn the optimal revenue, even when the time limit for the algorithm is augmented by any constant factor $c \geq 1$. We present 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 $1 + \lambda/k$, where λ denotes the ratio between the maximum and minimum distances in the graph, 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.

The Dial-a-Ride Problem (DARP) is well-studied with many variants including multiple vehicles, capacitated vehicles, and whether the requests are issued

ahead of time (offline) or on-the-fly (online). For a comprehensive overview of DARP please refer to the surveys *The dial-a-ride problem: models and algorithms* [8] and *Typology and literature review for dial-a-ride problems* [11].

In this work we study offline DARP on weighted graphs with a single server where each request has a source, destination, and priority. 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 priority. We assume uniform priorities so the goal is equivalent to maximizing the number of requests served within the deadline. A request's priority may represent the importance of serving the request in settings such as courier services. In more time-sensitive settings such as ambulance routing, the priority may represent the urgency of a request. In profit-based settings, such as taxi and ride-sharing services, a request's priority may represent the revenue earned for serving the request. The uniform priority variant in this work is useful for settings where all requests have equal priorities such as not-for-profit services that provide transportation to elderly and disabled passengers and courier services where deliveries are not prioritized. It also applies when revenue-based systems like Uber and Lyft are employed in dense geographic regions where requests have similar service times and therefore yield similar revenues. For the remainder of this paper, we will refer to the priority as "revenue," and to this revenue-maximization time-limited variant of the problem as RDARP.

One problem that is closely related to our work is the Prize Collecting Traveling Salesperson Problem (PCTSP) where the server earns a revenue (or prize) for every location it visits and a penalty for every location it misses, and the goal is to collect a specified amount of revenue while minimizing travel costs and penalties. PCTSP was introduced by Balas [4] but the first approximation algorithm, with ratio 2.5, was given by Bienstock et al. [5]. Later, Goemans and Williamson [10] developed a primal-dual algorithm to obtain a 2-approximation. A few years later, Awerbuch et al. [3] gave the first PCTSP approximation algorithms with polylogarithmic performance. Several years later, building off of the work in [10], Archer et al. [2] improved the ratio to $2 - \epsilon$, a significant result as the barrier of 2 was thought to be unbreakable [9].

To our knowledge, despite its relevance to modern-day transportation systems, aside from our work in [1], the revenue-maximizing time-limited version of DARP we investigate in this paper has not been previously studied in the offline setting. However, recently, Paul et al. [12, 13] studied a special case of our problem in which each request has the source equal to the destination; namely, they study the *budgeted* variant of PCTSP where the goal is to find a tour that maximizes the number of vertices 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. [6] has also presented the first constant-factor approximation algorithm for 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.

1.1 Our results

In Section 2 we begin by revisiting the Segmented Best Path (SBP) algorithm that we proposed in [7] for RDARP in the online setting. We show that SBP in the offline setting is a 4-approximation. We note that this is a tight bound as the lower bound of 4 we established in [7] for the online setting carries over to the offline setting.

In Section 3 we prove that no polynomial-time algorithm can be guaranteed to earn as much revenue as the optimal revenue, even when the time limit T for the algorithm is augmented by c for any constant $c \geq 1$.

In Section 4 we present our main result: *k-Sequence* (k-SEQ), a family of algorithms parameterized by k, for RDARP on weighted graphs and uniform revenues. 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. We let $\lambda = t_{max}/t_{min}$, where t_{min} and t_{max} denote the smallest and largest edge weights in the graph, respectively (note that in many real-world settings, λ may be viewed as a constant). We prove that k-SEQ has approximation ratio $(2 + \lceil\lambda\rceil/k)$. In Section 4.1 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., where the algorithm repeatedly serves the quickest request and therefore runs in polynomial time, the approximation ratio is $1 + \lambda$ and this is tight.

Finally, in Section 5, 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 $k > \lambda$.

2 Preliminaries

We formally define RDARP 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).[¶] 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. Since we focus here on the uniform-revenue variant of RDARP, each request in S can be considered as simply a pair

[¶]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. We further note that the input can be regarded as a metric space if the weights on the edges are expected to satisfy the triangle-inequality.

(s, d) where s is the source vertex or starting point of the request, and d is the destination vertex. We note that our problem is a generalization of budgeted PCTSP [12, 13]; while TSP 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 serve a request, the server must move from its current location x to s, then from s to d, and remains 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 set of requests C, where the minimum is taken over all permutations of the requests in C. We simplify it to *driveTime* when the set in consideration is clear from context.

With uniform revenues, the total revenue earned is equivalent to the number of requests served, and we thus use $|\textsc{alg}(I)|$ to denote the revenue earned by an algorithm ALG on an instance I of uniform-revenue RDARP, and we drop the I when the instance is clear from context. Similarly we use $|\textsc{opt}(I)|$ for the number of requests (or revenue) earned by the optimal solution OPT on instance I.

2.1 The SBP algorithm

Building upon prior work in DARP, we now analyze the SBP (SEGMENTED BEST PATH) algorithm proposed in [7] for the online variant of DARP with non-uniform revenues. Since our problem assumes uniform revenues, we unsurprisingly have a tighter upper bound, but note that the lower bound carries over.

Algorithm 1: Algorithm SEGMENTED BEST PATH (SBP) from [7]. Input: origin o, time limit $T > 0$, a complete graph G with maximum edge distance T/f, and a set of requests S given as source-destination pairs.

Let $t_1, t_2, \ldots t_f$ denote the time segments ending at times $T/f, 2T/f, \ldots, T$, respectively.
Let $i = 1$.
if f is odd **then**
    At t_1, do nothing. Increment $i = 2$.
end if
while $i < f$ **do**
    At the start of t_i, find the *max-revenue-request-sequence*, $\mathcal{R}$.
    if $\mathcal{R}$ is non-empty **then**
        Move to the source location of the first request in $\mathcal{R}$.
        At the start of t_{i+1}, serve requests in $\mathcal{R}$.
    else
        Remain idle for t_i and t_{i+1}
    end if
    Let $i = i + 2$.
end while

As described in Algorithm 1, the algorithm starts by splitting the total time T into f segments each of length T/f (where f is fixed and $1 < f < T$). At the start of a time segment, the server determines the *max-revenue-request-sequence*, $\mathcal{R}$, i.e. the maximum revenue-earning 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. Please refer to [7] for full details on how the algorithm finds $\mathcal{R}$.

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

Theorem 1. $|\textsc{opt}(S, T, o)| \leq 4|\textsc{sbp}(S, T, o)|$ *for any instance* (S, T, o) *of uniform-revenue* RDARP.

Proof. We first note that the lower bound instance in [7] for the online setting also applies to this offline setting; in that instance, SBP earns a revenue of no more than OPT/4.

For the upper bound, consider a schedule $\textsc{opt}_2$, 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 $\textsc{opt}_2$ 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 $\textsc{opt}_2$ is allowed one additional empty drive, we know $|\textsc{opt}_2(S, T, o)| \geq |\textsc{opt}(S, T, o)|$. We claim that $|\textsc{sbp}(S, T, o)| \geq 1/4|\textsc{opt}_2(S, T, o)|$, which implies that $|\textsc{sbp}(S, T, o)| \geq 1/4|\textsc{opt}(S, T, o)|$.

We proceed by strong induction on the number of time windows $w = \lceil f/2 \rceil$ where a time window is two consecutive time segments. For the base case let Q and R denote the set of requests served by $\textsc{opt}_2$ and SBP, respectively, in the first time window and let q and r denote their respective revenues. By the greedy nature of SBP, if $q = 1$, then $r = q$; if $q > 1$, since SBP serves requests during only every other time segment, then $r \geq q/2$ if q is even, and $r \geq (q-1)/2$ if q is odd. So if $w = 1$, then $r \geq q/4$, completing the base case.

For the inductive step, let P denote the path traversed by $\textsc{opt}_2$, let $p = |\textsc{opt}_2| \geq |\textsc{opt}|$ denote the number of requests served in P, and let u denote the first node $\textsc{opt}_2$ 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 $\textsc{opt}_2$ on $(S_{new}, T_{new}, o_{new})$ can move from o_{new} to u and serve these requests to earn revenue at least $p - (q + 1) - r$. By induction, on the smaller instance SBP will have revenue at

least $(p-q-1-r)/4$. Thus

$$|\textsc{sbp}(S,T,o)| = r + |\textsc{sbp}(S_{new},T_{new},o_{new})| \geq r + (p-q-1-r)/4 \\ \geq p/4 + (-q-1+3r)/4. \quad (1)$$

There are three cases for q and r.

1. Case: $q \geq 5$
   Then since $r \geq (q-1)/2$, from (1) we have:
   $|\textsc{sbp}(S,T,o)| \geq p/4 + (-q-1+3(q-1)/2)/4 \geq p/4 + (q/2-5/2)/4 \geq p/4$.
2. Case: $q \leq 4$ and $r \geq 2$
   From (1) we have: $|\textsc{sbp}(S,T,o)| \geq p/4 + (-4-1+6)/4 \geq p/4$.
3. Case: $q \leq 4$ and $r \leq 1$
   If $r = 0$, then $|\textsc{sbp}(S,T,o)| = |\textsc{opt}_2(S,T,o)| = 0$, so the theorem is trivially true, therefore, we assume $r = 1$. We first show by contradiction that every time window in $\textsc{opt}_2$'s schedule has fewer than 4 requests that end in that window. Suppose there is a window i in $\textsc{opt}_2$'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 $\textsc{opt}_2$ serves at least 1 request. We have $|\textsc{opt}_2(S,T,o)| < 4w'$ and $|\textsc{sbp}(S,T,o)| \geq \min(w,|S|) \geq (w')$, so $|\textsc{sbp}(S,T,o)| \geq 1/4|\textsc{opt}_2(S,T,o)|$. ■

3 c-Time Inapproximability

We now prove that no polynomial-time algorithm can be guaranteed to earn as much revenue as the optimal revenue, even when the time limit T for the algorithm is augmented by a constant factor. Let $I = (G,\sigma,T)$ denote an instance of RDARP, where G is the input graph, σ is the set of requests, and T is the time limit. We define ALG to be a *ρ-time-approximation* if ALG earns revenue at least OPT on the instance $(G,\sigma,\rho T)$. Portions of this proof are inspired by the NP-hardness proof of RDARP with uniform revenues provided in [1].

Theorem 2. *If $P \neq NP$, then there is no polynomial-time c-time-approximation to* RDARP *for any constant $c \geq 1$.*

Proof. We will show that a polynomial-time *c-time-approximation* to RDARP yields a polynomial-time decider for the directed Hamiltonian path problem (HAMPATH for short).

Given a directed HAMPATH input $G = (V,E)$ where $n = |V|$, we build an instance I for RDARP as follows. First, construct a complete graph G' with $2n+2$ nodes (see Figure 1; 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

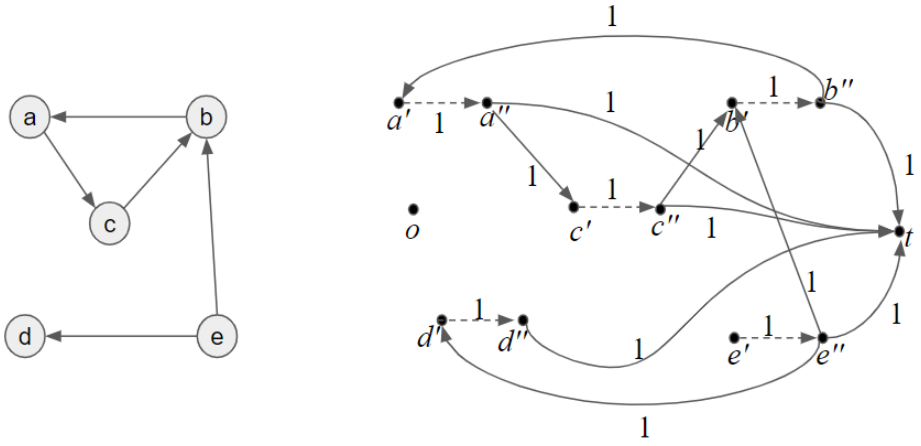

Fig. 1. An example instance G of HAMPATH where $n = 5$ (left), and the graph G' of the corresponding instance for RDARP where $T = 2n + 1$ (right). Four types of edges have distance 1: (1) all edges (v', v''), (2) for any $(u, v) \in G$, the edge (u'', v'), (3) for all nodes v' in G', the edge (o, v'), 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 $c \geq 1$.

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 $c \geq 1$. For σ, create a RDARP 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 RDARP 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 $I = (G', \sigma, T = 2n + 1)$.

By way of contradiction, we assume there is a *c-time-approximation* for RDARP called ALG and let $I' = (G', \sigma, cT = c(2n + 1))$ be an instance of RDARP where G' and σ are as described above. We claim ALG can be used as a polynomial-time decider for HAMPATH; 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 RDARP 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 RDARP request.

If either the sink or a single-primed node is the destination for the final RDARP request, the RDARP sequence must end with an edge-request. The alter-

nating structure ensures the RDARP 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 RDARP request, the RDARP sequence must end with a node request. The alternating structure ensures the RDARP sequence begins with an edge-request, and contains exactly n edge-requests and exactly n node requests. Thus, the RDARP sequence always contains n node requests. This ensures that the length n path in the original graph G includes all n vertices 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 $2n$ RDARP 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 $\text{ALG}(I') \geq 2n$. ■

4 k-Sequence upper bound

We now present k-Sequence (k-SEQ), our family of algorithms parameterized by k, for uniform-revenue RDARP (see Algorithm 2).

Algorithm 2: Algorithm k-Sequence (k-SEQ). Input: origin o, time limit $T > 0$, a complete graph $G^{\|}$, and a set of requests S given as source-destination pairs.

```
Set t := T.
while there are at least k requests left to serve 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 ≥ 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 ≤ k − 1 s.t. driveTime(C') ≤ t for some C' with |C'| = x.
If |C'| ≠ 0, serve C'.
```

[‖] We note again 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. We further note that the input can be regarded as a metric space if the weights on the edges are expected to satisfy the triangle-inequality.

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 determines how to serve all remaining requests optimally. If the remaining time is insufficient to serve any collection of k requests, the algorithm simply serves the largest set of requests that can be served within the remaining time. The algorithm runs in time polynomial in the number of requests but exponential in k since it must find all request sequences of length k.

Theorem 3. *k-SEQ is a $(2+\lceil\lambda\rceil/k)$-approximation for uniform-revenue* RDARP.

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 2). 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

**Note that when the graph is complete, t_{max} (t_{min}) is the maximum (minimum) distance over all pairs of nodes. When the graph is not complete, then using the pre-processing described in the previous footnote, we have that the distance between any two non-adjacent vertices is the shortest distance between those vertices, and t_{max} (t_{min}) is the maximum (minimum) distance over all of these distances.

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, $|k\text{-SEQ}(S,T,o)| = k + |k\text{-SEQ}(S_{new},T_{new},s_1)| \geq k + k/(2k+\lceil\lambda\rceil)(m-2k-\lceil\lambda\rceil) \geq k + km/(2k+\lceil\lambda\rceil) + k(-2k-\lceil\lambda\rceil)/(2k+\lceil\lambda\rceil) = k/(2k+\lceil\lambda\rceil)m$, completing the induction. ■

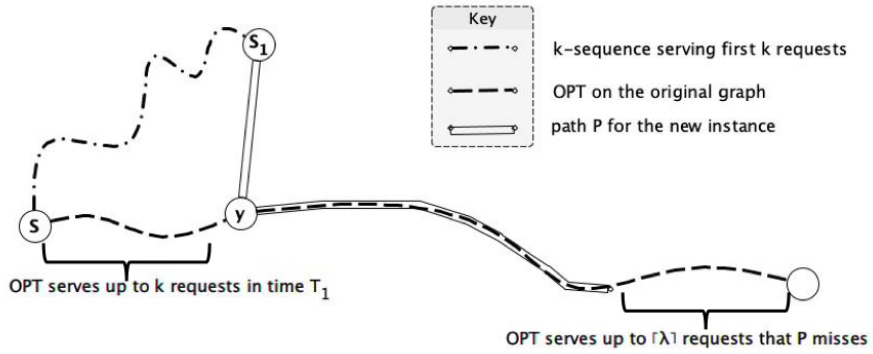

Fig. 2. An illustration of the paths taken by OPT and k-SEQ in Theorem 3. T_1 is the time needed for k-SEQ to serve its initial group of k requests, ending at s_1. The first node on the path of OPT after time T_1 is y. A feasible path P starting at time T_1 is from s_1 to y and then proceeds to the right. (It is possible for s_1 and y to be collocated.)

4.1 k-Sequence upper bound for large λ

We will now show for sufficiently large constant λ, the k-SEQ algorithm is a $(1+\lambda/k)$-approximation, which is a better ratio than the result obtained in Theorem 3 for sufficiently large λ. However, Theorem 3 remains better when $2+\lambda/k < k$.

Theorem 4. *For any instance I of uniform-revenue* RDARP, $|\text{OPT}(I)| < \max\{(1+\lambda/k)|k\text{-SEQ}(I)|+\lambda(k-1)/k+1, k|k\text{-SEQ}(I)|+k\}$. *I.e., when* $1+\lambda/k \geq k$, *we have*

$|\text{OPT}(I)| < (1+\lambda/k)|k\text{-SEQ}(I)| + \max\{\lambda(k-1)/k, k\}$, *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; since revenues are uniform, $q = \lceil n/k \rceil$. 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:

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

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 $\{r_{i_j}, \ldots, r_{i_j+k-1}\}$ 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:

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

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

$$\begin{aligned} qt_{max} &> x_J + \sum\{x_j : j \in \{1, \ldots, m\} \backslash \mathcal{I}\} + \sum\{y_j : j \in \{1, \ldots, m\} \backslash \mathcal{I}\} \\ &\geq \sum\{y_j : j \in \{1, \ldots, m\} \backslash \mathcal{I}\}. \end{aligned} \quad (5)$$

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. From Theorem 4 (in this section) and Theorem 5 (in the next section), we have 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 3 for an illustration of the lower bound).*

5 k-Sequence lower bound

In this section, we present lower bounds on k-SEQ; specifically, the lower bound is $\lambda+1$ for $k = 1$, shrinking to 2 for $k = \lambda$, and shrinking further towards $9/7$ for $k > \lambda$. We note that Theorem 5 matches the upper bound of Theorem 4 when $1 + \lambda/k \geq k$.

Theorem 5. *The approximation ratio of k-SEQ has lower bound $1 + \lambda/k$.*

Proof. Consider an instance (see Figure 3 for the case of $k = 1$) where there are two "paths" to follow, both a distance of λ away from the origin: one long chain of T requests, which is the path chosen by the optimal solution, and another "broken" chain that consists of k sequential requests at a time with a distance

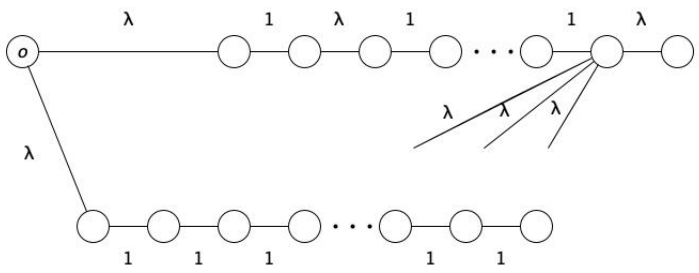

Fig. 3. The instance described in Theorem 5 when $k = 1$. Note that the graph is complete but only relevant edges are shown.

of λ from the end of each chain to any other request in the instance (similar to the instance of Figure 3, but instead of single requests, they now occur in chains of length k). 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 6. *The approximation ratio of k-SEQ has lower bound no better than 9/7 for any $k > \lambda$.*

Proof. Refer to the instance in Figure 4, with vertices: $o, a_0, a_1, \ldots, a_L$, $b_{k+\lambda+1}$, $\ldots, b_L$. The distances are: o is λ away from every vertex, $d(a_{i-1}, a_i) = 1$ for all $i = 1, \ldots, L$, and $d(a_i, b_i) = d(b_i, a_i) = 1$ for all $i = K + \lambda + 1, ..., L$. All other distances are λ.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 earns revenue $L+2(L-k-\lambda) = 3L-2\lambda-2k$ 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 $d(a_L, a_{L-2k}) = \lambda$. So the entire k-SEQ path then takes total time $(\lambda+k)L/k$ since there are L/k segments, and k-SEQ initially earns L revenue 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.

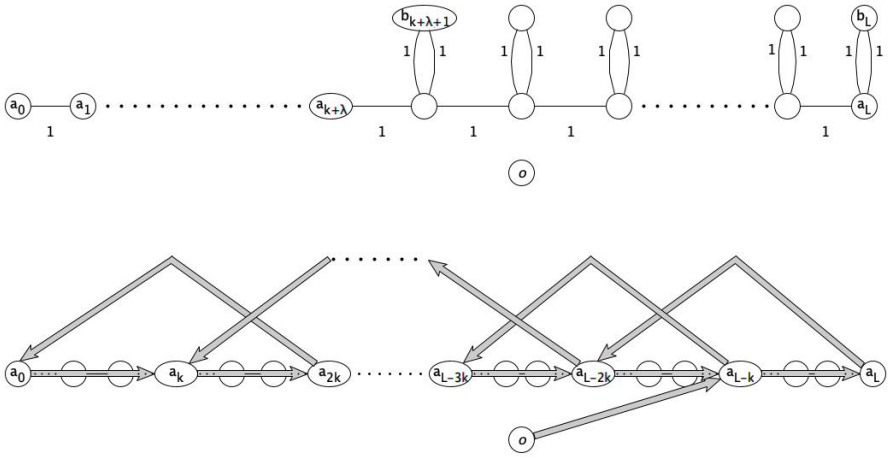

Fig. 4. Top: An instance where OPT earns no less than 9/7 the revenue of k-SEQ. Bottom: A depiction of the top instance illustrating the path taken by k-SEQ. k-SEQ starts at o, proceeds to serve the k requests from a_{L-k} to a_L, and then spends time moving to a_{L-2k} to serve the next collection of k requests, continuing similarly until the time limit. Note that in both figures the graph is complete but only relevant edges are shown.

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.

- k is even. Moving to the group (i.e. a sequence of k requests consisting of $k/2$ consecutive two-cycles)

$$\{(a_i, b_i), (b_i, a_i) \text{ for } i = j, \ldots, j + k/2 - 1\} \tag{6}$$

  for any j with $k + \lambda + 1 \leq j \leq L - k/2 + 1$ earns k revenue 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 earns $T'k/(\lambda + 3k/2 - 1)$ additional revenue in the remaining time T'; note for simplicity we can choose L so that $T'k$ is evenly divisible by $(\lambda + 3k/2 - 1)$.
- 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 (please see the Appendix for details).

In both cases, k-SEQ earns a total of $\frac{L+T' \cdot k}{(\lambda+3k/2-1)} = \frac{(7kL-2L-2k\lambda-4k^2)}{(3k+2\lambda-2)}$. 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 5 gives a lower bound of $1 + \lambda/k \geq 2$; so we have a lower bound of 9/7 for any k, λ.

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 $N + 1 < \lambda$. As a final remark, 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.

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. Awerbuch, B., Azar, Y., Blum, A., Vempala, S.: New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. SIAM Journal on Computing **28**(1), 254–262 (1998)
4. Balas, E.: The prize collecting traveling salesman problem. Networks **19**(6), 621–636 (1989)
5. 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)
6. 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)
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. Cordeau, J.F., Laporte, G.: The dial-a-ride problem: models and algorithms. Annals of Operations Research **153**(1), 29–46 (Sep 2007)
9. Goemans, M.X.: The prize-collecting TSP revisited (1998), SIAM Conference on Discrete Mathematics
10. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM Journal on Computing **24**(2), 296–317 (1995)
11. Molenbruch, Y., Braekers, K., Caris, A.: Typology and literature review for dial-a-ride problems. Annals of Operations Research **259**(1), 295–325 (2017)
12. 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)
13. 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)

6 Appendix

6.1 Proofs for Section 4

The following lemmas are used for the proof of Theorem 4.

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 that 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:*

$$i_1, \ldots, i_1 + k - 1, \quad \ldots, \quad i_{q-1}, \ldots, i_{q-1} + k - 1, \quad i_q, \ldots, i_q + \rho - 1 \tag{7}$$

such that $i_1 = 1$ and we have

$$\{i_j, ..., i_j + k - 1\} \cap \{\alpha_1, \ldots, \alpha_{k(j-1)}\} = \emptyset, \text{ for } 2 \leq j \leq q - 1, \tag{8}$$

$$\{i_j, ..., i_j + \rho - 1\} \cap \{\alpha_1, \ldots, \alpha_{k(j-1)}\} = \emptyset, \text{ for } j = q. \tag{9}$$

Proof. Set $i_1 = 1$. We now proceed to 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 ((8) and (9)) 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 (8) 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 $ch + dh - d + 1 = k^2(q-2) + 2k - 2 + 1 = k^2 q - 2k^2 + 2k - 1 = k^2 q - (k-1)^2 - k^2$ and $b = m - k - \rho \geq nk + k - k - \rho = (k(q-1) + \rho)k - \rho = k^2 q - k^2 + \rho(k-1)$. Since $b \geq k^2 q - k^2 + \rho(k-1)$ and $\rho(k-1) \geq 0 \geq -(k-1)^2$, we have $b \geq k^2 q - (k-1)^2 - k^2 = ch + dh - d + 1$ 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 $ch + dh - d + 1 = k^2(q - j - 1) + (j+1)k - (j+1) + 1 = k^2 q + j(-k^2 + k - 1) - k^2 + k = k^2 q - j(k-1)^2 - jk - k^2 + k$ whereas $b = m - \rho - jk \geq nk + k - \rho - jk = (k(q-1) + \rho)k - \rho - jk + k = k^2 q - k^2 + \rho(k-1) - jk + k$. It is now clear that $b \geq ch + dh - d + 1$, and the proof is complete. ■

6.2 Proof of odd case for Section 5

Specifically, k-SEQ from a_k 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 earns revenue k 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 earns revenue k in time $k + \lambda + (k-1)/2 - 1 = \lambda + 3k/2 - 3/2$. Together these two sequences earn revenue $2k$ 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 earns $T'2k/(2\lambda + 3k - 2)$ additional revenue in the remaining time T'. Note this is identical to the case of k even.