Revenue Maximization in Online Dial-A-Ride

Ananya Christman[1], Christine Chung[2], Nicholas Jaczko[3], Marina Milan[4], Anna Vasilchenko[5], and Scott Westvold[6]

1 **Dept. of Computer Science, Middlebury College, Middlebury, VT, USA**
achristman@middlebury.edu
2 **Dept. of Computer Science, Connecticut College, New London, CT, USA**
cchung@conncoll.edu
3 **Dept. of Computer Science, Middlebury College, Middlebury, VT, USA**
njaczko@middlebury.edu
4 **Dept. of Computer Science, Connecticut College, New London, CT, USA**
mmilan@conncoll.edu
5 **Dept. of Computer Science, Middlebury College, Middlebury, VT, USA**
avasilchenko@middlebury.edu
6 **Dept. of Computer Science, Middlebury College, Middlebury, VT, USA**
swestvold@middlebury.edu

Abstract

We study a variation of the Online-Dial-a-Ride Problem where each request comes with not only a source, destination and release time, but also has an associated revenue. The server's goal is to maximize its total revenue within a given time limit, T. We show that the competitive ratio is unbounded for any deterministic online algorithm for the problem. We then provide a 3-competitive algorithm for the problem in a uniform metric space and a 6-competitive algorithm for the general case of weighted graphs (under reasonable assumptions about the input instance). We conclude with an experimental evaluation of our algorithm in simulated settings inspired by real-world Dial-a-Ride data. Experimental results show that our algorithm performs well when compared to an offline version of the algorithm and a greedy algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, G.1.6 Optimization

Keywords and phrases online algorithms, dial-a-ride, competitive analysis, vehicle routing, metric space

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.1

1 Introduction

In the On-Line Dial-a-Ride Problem (OLDARP), a server travels in some metric space to serve requests for rides. The server has a *capacity* that specifies the maximum number of requests it can serve at any time. The server starts at a designated location of the space, the *origin*, and moves along the space to serve requests. Requests arrive dynamically and each request specifies a *source*, which is the pick-up (or start) location of the ride, a *destination*, which is the delivery (or end) location, and the release time of the request, which is the earliest time the request may be served. For each request, the server must decide whether to serve the request and at what time, with the goal of meeting some optimality criterion. In many variants preemption is not allowed, so if the server begins to serve a request, it must do so until completion. On-Line Dial-a-Ride Problems have many practical applications in

settings where a vehicle is dispatched to serve requests involving pick-up and delivery of people or goods. Important examples include ambulance routing, transportation for the elderly and disabled, taxi services, and courier services.

We study a variation of the Online-Dial-a-Ride Problem where in addition to the source, destination and release time, each request also has a priority and there is a time limit within which requests must be served. The server has unit capacity and the goal for the server is to serve requests within the time limit so as to maximize the total priority. A request's priority may simply 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 from serving the request. For the remainder of this paper, we will refer to the priority as "revenue," and to this variant of the problem as ROLDARP.

1.1 Related Work

The Online Dial-a-Ride problem was introduced by Feuerstein and Stougie [8] and several variations of the problem have been studied since. For a comprehensive survey on these and many other problems in the general area of *vehicle routing* see [11]. The authors of [8] studied the problem for two different objectives. One was to minimize the time to serve all requests and return to the origin (also known as *completion time*); the other was to minimize the average completion time of the requests (also known as *latency*). For minimizing completion time, they showed that any deterministic algorithm must have competitive ratio of at least 2 regardless of the server capacity. They presented algorithms for the cases of finite and infinite capacity with competitive ratios of 2.5 and 2, respectively. For minimizing latency, they proved that any algorithm must have a competitive ratio of at least 3. They also presented a 15-competitive algorithm for the infinite capacity problem on the real line.

Ascheuer et al. [1] studied minimizing total completion time for OLDARP with multiple servers and capacity constraints and presented a 2-competitive algorithm for this problem. Jaillet and Wagner [10] considered a version of OLDARP where each request consists of one or more locations, and precedence and capacity requirements for the locations. To serve a request the server must visit each location, while satisfying the requirements. The authors provided a non-polynomial 2-competitive algorithm for minimizing completion time.

The Online Traveling Salesperson Problem (OLTSP), introduced by Ausiello et al. [2], is a special case of OLDARP where for each request the source and destination are the same location. Krumke [13] studied both OLDARP and OLTSP for the uniform metric space. Their objective was to minimize the maximum *flow time*, that is the difference between a request's release and service times. They proved that no competitive algorithm exists for OLDARP and gave a 2-competitive algorithm to solve OLTSP.

Within the last five years many heuristic approaches have emerged as means for tackling practical variations of Dial-a-Ride problems; for example, see [15, 3, 12, 14, 9]. On the other hand, provably competitive results on OLDARP have been sparse within the last five years; we are unaware of any such work, other than [4]. For Revenue-maximizing OLDARP (ROLDARP), [4] showed that no deterministic online algorithm can be competitive on graphs with non-uniform edge weights. They therefore focused on the uniform metric and presented a greedy 2-competitive algorithm for this problem. They also considered two variations of this problem: (1) the input graph is complete bipartite, and (2) there is a single node that is the source for every request, and presented a 1-competitive algorithm (optimal to within an additive factor) for the former and an optimal algorithm for the latter [4].

1.2 Our Results

In this work we begin by proving that the offline version of ROLDARP is NP-hard. We then present a greedy algorithm called BestPath (BP) for ROLDARP in the uniform metric. Although a 2-competitive algorithm has already been found [4], we show that a "smarter" algorithm like BP is in fact only 3-competitive. The idea of the BP algorithm also forms the basis for our main result: the algorithm we present in Section 4, Segmented Best Path (SBP), which we show is 6-competitive for ROLDARP on weighted graphs, provided that the edge weights are bounded by T/f where T is the time limit and $1 < f < T$, and that the revenue earned by the optimal offline solution in the last $2T/f$ time units is bounded by a constant. We show that the competitive ratio is unbounded for any deterministic online algorithm for the problem, unless we assume edge weights are bounded and discount the revenue earned by OPT in the last T/f time units. We also give a lowerbound of 4 on the competitive ratio of SBP. As far as we know, this is the first work that presents a competitive algorithm for ROLDARP on weighted graphs. Finally, in Section 5, we present some experimental results for the SBP algorithm in simulated settings inspired by real-world Dial-a-Ride data. Our experimental results show that SBP performs well when compared to an offline version of SBP and a simple greedy algorithm.

2 Preliminaries

Formally, we define the ROLDARP problem 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 weight $w_{u,v} > 0$. (We note that in fact 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 weight 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.) 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 sequence of requests, σ, that are dynamically issued to the server.

Each request is of the form (s, d, t, p) where s is the source node, d is the destination, t is the time the request is released, and p is the revenue (or priority) earned by the server for serving the request. The server does not know about a request until its release time t. To serve a request, the server must move from its current location x to s, then from s to d. The total time for serving the request is equal to the length of the path from x to d. We assume the earliest time a request may be released is at $t = 0$.

For each request, the server must decide whether to serve the request and if so, at what time. A request may not be served earlier than its release time and at most one request may be served at any given time. Once the server starts serving a request, it must serve the request until completion (i.e. preemption is not allowed). The goal for the server is to serve requests within the time limit so as to maximize the total earned revenue.

We use R-DARP to refer to the offline version of ROLDARP, which knows all requests from the start, at time 0.

▶ **Theorem 1.** *R-DARP is NP-hard.*

Proof. R-DARP-D, the decision version of R-DARP, outputs YES on an input instance if and only if there is a set of requests that can be served to yield a revenue that is greater than or equal to a given value k.

We proceed by reducing the Knapsack problem to R-DARP-D. The Knapsack problem is defined as follows: We are given a set of items S, where item $i \in S$ has a positive integer weight w_i and a positive integer value v_i, and a knapsack weight limit W and some value c. We must determine whether there is a subset of S that has total weight at most W and value at least c.

From an instance of 01-Knapsack we build an instance of R-DARP-D as follows. Create an empty graph G. For each item $i \in S$, we will make a request r_i with a release time of 0 and an arbitrary source s_i and destination d_i that have not been used by another request. Let the revenue $p_i = v_i$ and add s_i and d_i to G as vertices with an undirected edge of weight w_i between them.

After we have created a request as above for each item in S, we add one more vertex, o, to serve as the origin for the server. We then add an edge of weight $\frac{1}{n}$ between any pair of vertices that does not yet have an edge between them. We set $T = W + 1$ and $k = c$.

If there is a feasible set of requests with total revenue at least k, serving the set of requests requires at least one empty move (because no two requests share an endpoint and none start at the origin). So, the empty moves will take up at least $\frac{1}{n}$ time units. The lengths of the edges that represent requests are all positive integers, so the total time to serve the requests (not including the time to move between requests) is at most $T - 1$ time units. Since $W = T - 1$ and the lengths of the requests were determined by the weights of the items in S, the requests correspond to a set of items S' with total weight at most W. The revenues of the requests were determined by the values of the items in S. Thus, the items in S' have total value at least $c = k$.

If there is a subset S' of S whose items have total value at least c and a total weight at most W, then it will take at most $W = T - 1$ time units to serve the requests that correspond to those items, not including the time to move from one request to another. Each move step will take at most $1/n$ and there will be at most n of these steps, so the total amount of time required to move will be no more than 1. Therefore, all of the requests can be fulfilled in T time units or less. The revenue yielded by each request is equal to the value of a corresponding item in S', so the total revenue will be at least $k = c$. ◀

We let OPT denote an optimal offline algorithm, as in, an algorithm that given any sequence of requests will serve the requests that return the maximum possible revenue for that input. Given an input graph G, a sequence $\sigma = r_1, \ldots r_m$ of requests and an algorithm ALG, we denote $\text{ALG}(G, \sigma)$ as the total revenue earned by ALG from σ on G. We say that ON is γ-*competitive* if there exists $\gamma \geq 1, b \geq 0$ such that for all σ:

$$\text{OPT}(G, \sigma) \leq \gamma \cdot \text{ON}(G, \sigma) + b\,. \tag{1}$$

In [4] it was shown that no deterministic algorithm can be competitive when edge weights are non-uniform and when revenues can take on any arbitrarily large value. In particular, it was shown that in Equation 1, if b is set to the last revenue earned by OPT, p_{last}, then $\gamma \cdot \text{ON}(G, \sigma) + b < \text{OPT}(G, \sigma)$ for any $\gamma \geq 1$. We now show that the non-competitiveness is due to arbitrarily large edge weights alone. In other words, if edge weights may be arbitrarily large, then regardless of revenue values, no deterministic algorithm can be competitive.

For the remainder of this work, we use the terminology "algorithm A *serves* (or *has served*) request r at time t" to indicate that A *begins* serving request r with source s and destination d at time t and *completes* serving r at time $t + w_{s,d}$.

▶ **Lemma 2.** *There is no deterministic algorithm for ROLDARP that is γ-competitive, for any $\gamma \geq 1$.*

Proof. The adversary will release requests in such a way that regardless of the behavior of the online algorithm, the optimal offline solution will earn more than γ times the revenue earned by any online algorithm, for any value of $\gamma \geq 1$. The instance will begin with two requests, and if the online algorithm chooses to serve either one, the adversary will release a large number of requests the online algorithm is unable to serve.

Let G denote a complete graph with three nodes s, x, and y, where s is the origin, $w_{s,x} = c$ where $c = 2\gamma + 3$, with $\gamma \geq 1$, $w_{s,y} = 1$ and $w_{x,y} > c$, let $T \geq 2c$ denote the time limit. The adversary will release two requests: $r_1 = (s, x, T - 2c, p)$ and $r_2 = (x, s, T - c, p)$, with $p > 0$. There are two cases for the online algorithm, ON:

Case 1: on serves neither of these requests. Since there is enough time for OPT to serve both requests, $\text{OPT}(G, \sigma) = 2p$ while $\text{ON}(G, \sigma) = 0$. For $b = p_{last} = p$ we have $0\gamma + p < 2p$ for any $\gamma \geq 1$ and $p > 0$, so ON is not competitive.

Case 2: on serves at least one of these requests. Suppose ON starts serving a request at time t. Then the adversary will release c requests where $c/2$ of the requests are $(s, y, t + 1, p)$ and the other $c/2$ are $(y, s, t + 1, p)$.

We will now show that ON will not have enough time to serve any of these requests but OPT will serve all of them and earn revenue $\text{OPT}(G, \sigma)$ such that for any $\gamma \geq 1$, $\gamma \cdot \text{ON}(G, \sigma) + p_{last} < \text{OPT}(G, \sigma)$. There are two sub-cases:

1. ON serves r_1 (and possibly r_2). The earliest ON may serve any of r_1 and r_2 is at $T - 2c$ and in particular, it may serve only r_1 at this time. It will complete r_1 no sooner than time $T - c$. To serve any of the new requests, it must move to either s or y from x. It will arrive at either s or y no sooner than T and therefore will not have enough time to complete any of these requests.
2. ON serves only r_2. The earliest ON may serve r_2 is at $T - c$ and it will complete it no sooner than time T and will therefore not have enough time to complete any of the new requests.

In both cases ON earns at most $2p$.

Note that the latest that ON may serve either r_1 or r_2 is at $T - c$ (so $t \leq T - c$). Therefore $t + 1 \leq T - c + 1$ and for every time unit from $t + 1 \leq T - c + 1$ to T, OPT earns revenue p, so OPT earns total revenue at least $(T - (T - c + 1))p = (c - 1)p$, so $\text{OPT}(G, \sigma) > (c - 2)p$. For $c = 2\gamma + 3$ we have:

$$\text{OPT}(G, \sigma) > (2\gamma + 1)p = 2\gamma p + p \geq \gamma \cdot \text{ON}(G, \sigma) + p = \gamma \cdot \text{ON}(G, \sigma) + p_{last} \tag{2}$$

for all $\gamma \geq 1$ and $p_{last} > 0$. ◀

Since we have now established that no deterministic online algorithm can be competitive if edge weights are not bounded, we assume for the remainder of this work that **all edge weights are no more than T/f for some $1 < f < T$.**

However, even with this restriction we find that no deterministic online algorithm can serve the requests served by OPT during the last T/f time units. Hence the competitive ratio of any algorithm is still unbounded unless b is set to the revenue earned by OPT in last T/f time units.

▶ **Lemma 3.** *If the maximum edge weight is T/f for some $1 < f < T$, no deterministic online algorithm can serve the requests served by* OPT *during the last T/f time units.*

Proof. Let ON denote a deterministic online algorithm. There are two cases:

Algorithm 1: Algorithm BEST PATH (BP). Input is complete unit weight graph G and time limit T.

```
1:  for t = 1 to T do {at each unit of time, do the following}
2:    if a request was served at time t − 1 then
3:      define P to be the currently available request-path with the highest score.
4:      if P does not start at the current server location then
5:        move server to the start of P.
6:      else
7:        serve the first request on the path P
8:      end if
9:    else {no request was served at time t − 1}
10:     serve the first request of P {which was determined in iteration t − 1}
11:   end if
12: end for
```

1. At time $T-(T/f)$, ON is at (or moving toward) a node s such that there is an edge (s,x) where $w_{s,x}=T/f$. Then the adversary releases the request $(x,s,T-(T/f),p)$ for some $p>0$. ON will not be able to serve the request since an online server will arrive at x no sooner than time T, however an optimal algorithm could serve the request.
2. At time $T-(T/f)$, ON is at (or moving toward) a node s such that there is no edge (s,x) where $w_{s,x}=T/f$. (So its weight is strictly less than T/f.) Then let u and z denote two nodes such that $w_{u,z}=T/f$ and $w_{s,u}=\epsilon$ for $\epsilon>0$. The adversary releases the request $(u,z,T-(T/f),p)$ for some $p>0$. ON will not be able to serve the request since an online server will arrive at u no sooner than $T-(T/f)+\epsilon$ and would not be able to complete the request by time T, however an optimal algorithm could serve the request. ◀

3 The Uniform Metric

In this section we provide a greedy 3-competitive algorithm called Best Path (BP) for ROLDARP on a complete graph with unit edge weights (or in a uniform metric). We use the term "empty move" to refer to when an algorithm moves the server from one point in the metric space to another (expending one unit of time) without serving a request. We use the term *request-path* to refer to a path of "connected" requests (with no empty moves required in between) that are currently outstanding (released but not yet served).

The Greatest Revenue First (GRF) algorithm of [4] simply serves the request with the highest revenue at every other time unit (spending the time units in between either standing still or on empty moves. In contrast, the BP algorithm takes into account the time it saves when serving a contiguous sequence of connected requests by finding and choosing the request-path that has the highest revenue per time unit, which we refer to as the "score". Specifically, we let $score(P)$ of a request-path P be the total revenue of the requests in P divided by the time it takes to complete P, including the time it takes to move from the current server location to the start of P.

Although GRF was already shown in [4] to be 2-competitive, here we present the 3-competitiveness of the BP algorithm because (1) it is surprising that the simpler more "naive" GRF algorithm has a better competitive ratio, and (2) the Segmented Best Path (SBP) algorithm, which we present in Section 4 for weighted graphs, is a hybrid of GRF and BP, and (3) BP out-performs GRF in experimental simulations. We note that step 3 of the algorithm

may not be achieved in strict polynomial time (see the run-time discussion of the analogous step in our SBP algorithm of Section 4 for more details). We defer the proof of the following theorem, along with our experimental results on the BP algorithm to the full version of this work.

▶ **Theorem 4.** *The Algorithm* BP *is 3-competitive, and this is tight.*

4 Weighted Graphs

4.1 The algorithm

We now describe our online algorithm for weighted graphs (see Algorithm 2 for details). The algorithm splits the total time T into f segments of length T/f where $1 < f < T$. At the start of every other time segment it considers all the unserved requests that have been released, finds all sets of requests that can be served within one time segment (i.e. within T/f amount of time), and determines the set that yields the maximum total revenue. We refer to this as the *max-revenue-request-set*. It then moves to the source of the first request in this set and at the start of the next time segment, serves the requests in this set.

To find the max-revenue-request-set, the algorithm maintains a directed auxiliary graph, G', which we refer to as the *request graph*, to keep track of unserved requests. More specifically, at the beginning of every other time segment the algorithm does the following:

1. For every unserved request $r = (s, d, t, p)$, add a directed edge (s, d) to G' (so parallel edges are allowed) with weight equal to its corresponding weight in G, and label this new edge (s, d) with the revenue p. We refer to edges added to G' in this step as *request-edges*.
2. Add a directed edge labeled with revenue 0 from the destination node of each request-edge to the source node of every other request edge. The weights of these edges are the same as in G.
3. For every pair of nodes u, v in G', find all paths of length at most T/f from u to v.
4. For each path P found in the previous step, let π_P denote the sum of revenues earned from the requests that correspond to the request-edges in P.
5. The *max-revenue-request-set* is the path that has max π_P value.

The bottleneck in the time complexity of the algorithm occurs in step 3 of the above subroutine. We must enumerate all paths in G' of length at most T/f, and the number of possible paths is exponential in the size of G', which is determined directly by the number of outstanding requests in the current time segment. In many real world settings, one can expect the size of G' to be small relative to the size of G. And in settings where T/f is small, the run time is further minimized, making it feasible to execute efficiently in many plausible settings.

4.2 Lowerbound

We prove that SBP has a competitive ratio no better than 4 by providing an instance where the ratio approaches 4 as T grows (with the additive term b equal to the revenue earned by OPT within the last two time segments).

▶ **Theorem 5.** *If* SBP *is c-competitive for ROLDARP, then* $c \geq 4$.

Proof. Consider an instance (G, σ) of ROLDARP as follows. For some even f, there are T requests released at time 0 and each request requires $(T/(2f)) + 1$ time to serve and has priority/revenue 1. In the first $f - 2$ time segments, OPT serves $\frac{T-(2T/f)}{(T/(2f))+1}$ requests and earns

Algorithm 2: Algorithm Segmented Best Path (SBP). Input is complete graph G with time limit T and maximum edge weight T/f.

Let $t_1, t_2, \ldots t_f$ denote the time segments ending at times $T/f, 2T/f, \ldots, T$, respectively.
if f is odd **then**
  At t_1, do nothing.
  At the start of every t_i for even $i \geq 2$ find the *max-revenue-request-set* and move to the source location of the first request in this set. Denote this request set as R. If no unserved request sets exist, do nothing.
  At the start of every t_i for odd $i \geq 3$, serve request set R (if it exists) from the previous step.
end if
if f is even **then**
  At the start of every t_i for odd $i \geq 1$ find the *max-revenue-request-set* and move to the source location of the first request in this set. Denote this request set as R. If no unserved request sets exist, do nothing.
  At the start of every t_i for even $i \geq 2$, serve request set R (if it exists) from the previous step.
end if

this amount of revenue. 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 and earns this amount of revenue. We have:

$$\frac{\text{OPT}(G,\sigma)}{\text{SBP}(G,\sigma)} \geq \frac{\left(\frac{T-(2T/f)}{(T/(2f))+1}\right)}{f/2} \geq \frac{4T(f-2)}{(T+2f)f} \tag{3}$$

◀

For any $f > 2$, as $T \to \infty$, the above expression approaches 4.

4.3 Upperbound

We analyze the revenue earned by Algorithm 2 by considering the time segments in pairs. We refer to each pair of consecutive time segments as a time window, so if there are f time segments, there are $\lceil f/2 \rceil$ time windows. Note that the last time window may have only one time segment.

For notational convenience, we consider a modified version of the SBP schedule, that we refer to as SBP$'$, which serves exactly the same set of requests as SBP, but does so one time window earlier. Specifically, if SBP serves a set of requests during time window $i \geq 2$, SBP$'$ serves this set during time window $i-1$ (so SBP$'$ ignores the set served by SBP in window 1). We note that the schedule of requests served by SBP$'$ may be infeasible, and that it will earn at most the amount of revenue earned by SBP.

For time window $i = 1 \ldots \lceil f/2 \rceil - 1$, let S'_i be the set of requests served by SBP$'$, which by definition will serve requests during one of the two time segments of window i, and will use the other time segment of window i to move.

Let $rev(S)$ denote the revenue earned from a set of requests S. Let $S^*(t_j)$ denote the set of requests served by OPT in time segment t_j and let S^*_i denote the set of requests served by OPT during the time segment of window i with greater revenue, i.e. $S^*_i =$

$\arg\max\{rev(S^*(t_{2i-1})), rev(S^*(t_{2i}))\}$. Note this set may include a request that was started in the prior time segment, as long as it was completed in the time segment of S^*_i.

We will first show that over the time windows $i = 1 \dots \lceil f/2 \rceil - 1$, OPT earns no more than three times the amount of revenue earned by SBP′.

▶ **Lemma 6.** $\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S^*_i) \leq 3 \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S'_i)$.

Proof. Let H denote the chronologically ordered set of time windows w where $rev(S^*_w) > rev(S'_w)$, and let h_j denote the jth time window in H. We refer to each window of H as a window with a "hole," in reference to the fact that SBP′ does not earn as much revenue as OPT in these windows.

In each window h_j there is some amount of revenue that OPT earns that SBP′ does not. In particular, there must be a set of requests that OPT serves in window h_j that SBP′ does not serve in h_j. Note that if this set is not available for SBP′ in h_j then SBP′ must have served it in some previous windows. Let $S^*_{h_j} = A_j \cup B_j \cup C^*_j$, where A_j is the subset of requests served by both OPT and SBP′ in h_j, B_j is the subset of requests served in h_j by OPT, but served previously by SBP′, and C^*_j is the subset of OPT's requests available for SBP′ but SBP′ chooses not to serve.

Our plan is to build an infeasible schedule $\overline{\text{SBP}}$ that will be similar to SBP′, but contain additional "copies" of some requests such that no windows of $\overline{\text{SBP}}$ contain holes. We will make no more than 3 copies of each request in SBP′, so ultimately $\overline{\text{SBP}}$ will have revenue at most 3 times that of SBP′.

We first initialize $\overline{\text{SBP}}$ to have the same schedule of requests as SBP′. We then add additional requests to h_j for each $j = 1 \dots |H|$, based on $S^*_{h_j}$. Recall that in our accounting of the revenue of OPT we have allowed requests to be part of a time segment even if OPT began serving them in the previous time segment (see the definition of S^*_i above.) So the set $S^*_{h_j}$ may not fit within a single time segment. We consider two cases based on $S^*_{h_j}$:

1. The set $S^*_{h_j}$ can be served within one time segment. In this case we know that there must be a non-empty subset of $S^*_{h_j}$, B_j, that is not available for SBP′ to serve in h_j (because if it were available SBP′ would serve it or something better, due to its greediness), so SBP′ must have served B_j in some set W_j of previous time windows. Specifically, a time window w is in W_j if and only if a request from B_j was served in w by SBP′.
   Let us refer to the set of requests served by SBP′ in h_j as $S'_{h_j} = A_j \cup C_j$ for some set of requests C_j. Notice that if $S^*_{h_j} = A_j \cup B_j \cup C^*_j$ can be executed within a single time segment, then $rev(C_j) \geq rev(C^*_j)$, again by greediness of SBP′.
   In this case, in $\overline{\text{SBP}}$ we add an additional "copy" of the set B_j to h_j. So the requests in B_j each appear twice in the $\overline{\text{SBP}}$ schedule. Now, h_j will no longer be a hole in SBP since
   $$rev(S^*_{h_j}) = rev(A_j) + rev(B_j) + rev(C^*_j) \leq rev(A_j) + rev(B_j) + rev(C_j) = rev(\overline{S}_{h_j}),$$
   where $\overline{S}_{h_j}$ is the set of requests served by $\overline{\text{SBP}}$ in h_j.
2. The set $S^*_{h_j}$ cannot be served within one time segment. Let k be the index of the time segment corresponding to $S^*_{h_j}$. In this case, OPT must have begun serving a request of $S^*_{h_j}$ in time segment t_{k-1} and completed this request in time segment t_k. Let us use r^* to denote this request that "straddles" the two time segments. In addition to the copy of B_j as in Case 1 above, for this case we will also add to $\overline{\text{SBP}}$ a copy of another set of requests. Again, let us refer to the set of requests served by SBP′ in h_j as $S'_{h_j} = A_j \cup C_j$ for some set of requests C_j. There are two subcases depending on whether $r^* \in C^*_j$ or not.
   a. $r^* \in C^*_j$. In this case we have $rev(C_j) \geq \max\{rev(r^*), rev(C^*_j \setminus \{r^*\})\} \geq \frac{1}{2} rev(C^*_j)$.

So, in addition to the copy of B_j as in Case 1, we also add a copy of the set C_j to the $\overline{\text{SBP}}$ schedule, so it serves two copies of C_j in h_j. Note that for $\overline{\text{SBP}}$, h_j will no longer be a hole since

$$rev(S^*_{h_j}) = rev(A_j)+rev(B_j)+rev(C^*_j) \leq rev(A_j)+rev(B_j)+2\cdot rev(C_j) = rev(\overline{S}_{h_j})\,,$$

where $\overline{S}_{h_j}$ is the set of requests served by $\overline{\text{SBP}}$ in h_j.

b. $r^* \notin C^*_j$. In this case C^*_j can be served within one time segment but SBP$'$ chooses to serve $A_j \cup C_j$ instead. So we have $rev(A_j) + rev(C_j) \geq rev(C^*_j)$, therefore we know either $rev(A_j) \geq \frac{1}{2} rev(C^*_j)$ or $rev(C_j) \geq \frac{1}{2} rev(C^*_j)$. In the latter case, we can do as we did in sub-case (a) above and add a copy of the set C_j to the $\overline{\text{SBP}}$ schedule in window h_j, to get $rev(S^*_{h_j}) \leq rev(\overline{S}_{h_j})$, as above. In the former case, we instead add a copy of A_j to the $\overline{\text{SBP}}$ schedule in window h_j. Then again, for $\overline{\text{SBP}}$, h_j will no longer be a hole, since this time

$$rev(S^*_{h_j}) = rev(A_j)+rev(B_j)+rev(C^*_j) \leq 2\cdot rev(A_j)+rev(B_j)+rev(C_j) = rev(\overline{S}_{h_j}).$$

For each window w not in H, we already have the property that $rev(S^*_w) \leq rev(S'_w)$, by the definition of H. Therefore, in every window for $i = 1 \ldots \lceil f/2 \rceil - 1$, $\overline{\text{SBP}}$ earns at least as much revenue as OPT. Let $rev(\overline{S}_i)$ denote the revenue earned by $\overline{\text{SBP}}$ in window i. We have:

$$\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(\overline{S}_i) \geq \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S^*_i)\,. \tag{4}$$

Also note that $\overline{\text{SBP}}$ serves each request no more than three times, as it makes at most two additional copies of each request in SBP$'$ (and this is only in the event the same request in C_j was also copied to some later set B_i). Therefore:

$$\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(\overline{S}_i) \leq 3 \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S'_i)\,. \tag{5}$$

Combining 4 and 5 yields:

$$\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S^*_i) \leq 3 \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S'_i). \tag{6}$$

◀

▶ **Theorem 7.** *If the revenue earned by* OPT *in the last two time segments is bounded by some constant, then* SBP *is 6-competitive, i.e., if* $rev(S^*(t_f)) + rev(S^*(t_{f-1})) \leq c$ *for some constant* c*, then*

$$\sum_{j=1}^{f} rev(S^*(t_j)) \leq 6 \sum_{j=1}^{f} rev(S(t_j)) + 2c\,. \tag{7}$$

Proof. By Lemma 6 we know that $\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S^*_i) \leq 3 \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S'_i)$. Let S_i denote the set of requests served by SBP during time window i. Then since SBP$'$ earns at most the revenue of SBP, Lemma 6 implies:

$$\sum_{i=1}^{\lceil f/2 \rceil} rev(S^*_i) \leq 3 \left(\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S_i) \right) + rev(S^*(t_f)) + rev(S^*(t_{f-1}))\,. \tag{8}$$

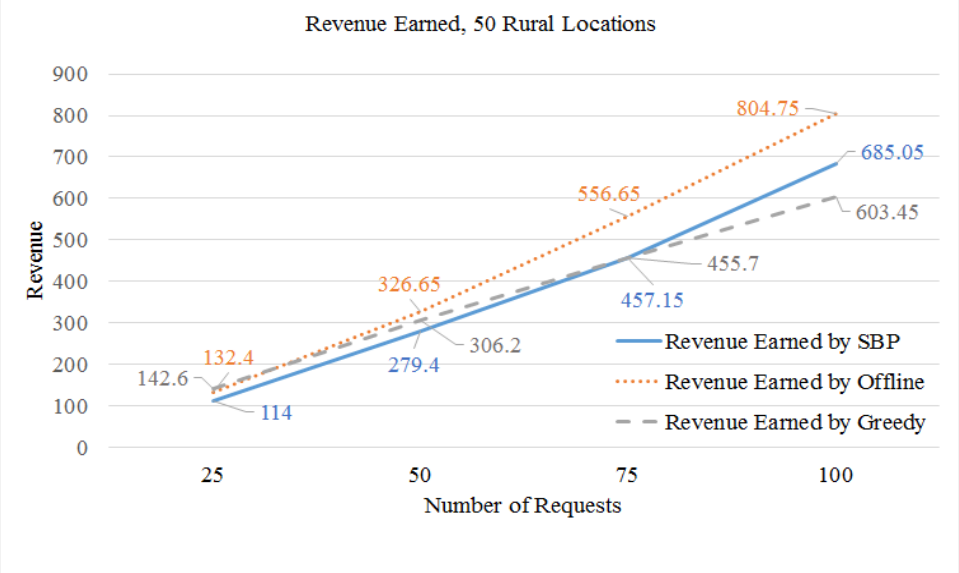

Figure 1 Revenue earned for rural setting.

By the definition of SBP, we know:

$$\sum_{i=1}^{\lceil f/2 \rceil} rev(S_i) = \sum_{j=1}^{f} rev(S(t_j)) \tag{9}$$

and by the definition of S_i^*, we have:

$$\sum_{j=1}^{f} rev(S^*(t_j)) \leq 2 \sum_{i}^{\lceil f/2 \rceil} rev(S_i^*)\,. \tag{10}$$

Equations 8, 9, and 10 imply:

$$\sum_{j=1}^{f} rev(S^*(t_j)) \leq 6 \left(\sum_{j=1}^{f} rev(S(t_j)) \right) + 2rev(S^*(t_f)) + 2rev(S^*(t_{f-1})). \quad \blacktriangleleft$$

5 Experimental Results

To evaluate the performance of the SBP algorithm, we simulated three realistic Online Dial-a-Ride systems. Our experimental settings were informed by data we retrieved from real-world Dial-a-Ride systems ([5, 6, 7]) and reflected three Dial-a-Ride environments: rural, suburban, and urban. The problem inputs varied based on the environment type.

We now describe the three settings more specifically. In each setting, a time unit is 10 minutes, so there are 6 time units per hour. The graph is complete and contains 50 nodes where each is equally likely to be a source or destination of a request (but the same node cannot be both the source and destination). The origin is randomly chosen from the set of nodes. The release times of requests are non-integral values uniformly distributed from $t = 0$ to $t = T - T/f$ (since no deterministic online algorithm can compete with requests served by OPT during during the last T/f time units; see Lemma 3). Request priorities are

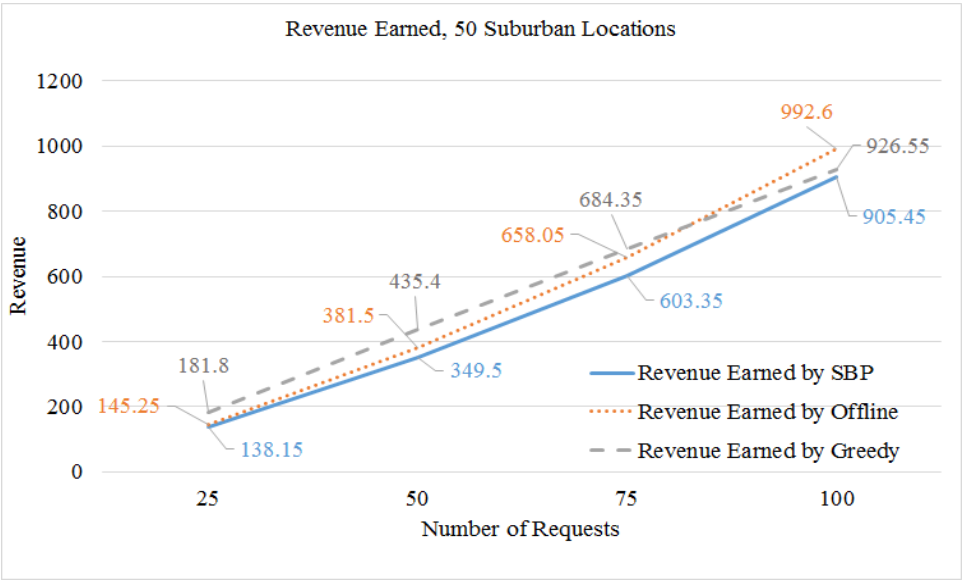

Figure 2 Revenue earned for suburban setting.

integral values uniformly distributed from 1 to m where m is the number of requests (we test $m = 25, 50, 75$, and 100). A request's priority represents the urgency of the request (i.e. transport to a pharmacy is more urgent than to a shopping mall).

1. Rural: Time spans 9 hours (from 8:00am to 5:00pm), so $T = 54$. The maximum distance between locations is 60 minutes (i.e. 6 time units), so $T/f = 6$ and $f = 9$. Edge weights are chosen uniformly at random from the interval $[1, T/f]$.
2. Suburban: Time spans 9 hours (from 8:00am to 5:00pm), so $T = 54$. The maximum distance between locations is 45 minutes (i.e. 4.5 time units), so $T/f = 4.5$ and $f = 12$. Edge weights are chosen uniformly at random from the interval $[1, T/f]$.
3. Urban: Time spans 11 hours (from 6:00am to 7:00pm), so $T = 66$. The maximum distance between locations is 20 minutes (i.e. 2 time units), so $T/f = 2$ and $f = 33$. Edge weights are chosen uniformly at random from the interval $[0.5, T/f]$.

Since R-DARP is NP-hard to solve optimally (see Theorem 1), we instead compare our algorithm, SBP (see Algorithm 2), to an offline version of SBP, as well as to a basic greedy algorithm. Like SBP, the offline-SBP algorithm serves requests in time segments – i.e. it uses one time segment to determine and move to the maximum-revenue-request-set and the next time segment to serve this set. However, unlike SBP, this offline algorithm learns of all the requests at the start of time ($t = 0$) and is therefore able to make a more informed decision about which requests to serve. In contrast, the greedy algorithm does not break time into segments and instead simply moves to and serves the outstanding request whose revenue is highest (similar to the GRF algorithm of [4]).

Figures 1, 2, and 3 show the results of the experimental simulations. The graphs show that for all three settings SBP is competitive (colloquially speaking) with the offline and greedy algorithms. Specifically, in the rural setting, SBP earns approximately 82-86% of the revenue earned by the offline algorithm and it earns more revenue than the greedy algorithm when the number of requests is high (100).

Depending on the total number of released requests, SBP earns between 91-95% of the revenue earned by the offline algorithm in the suburban setting, and between 88-90% in the

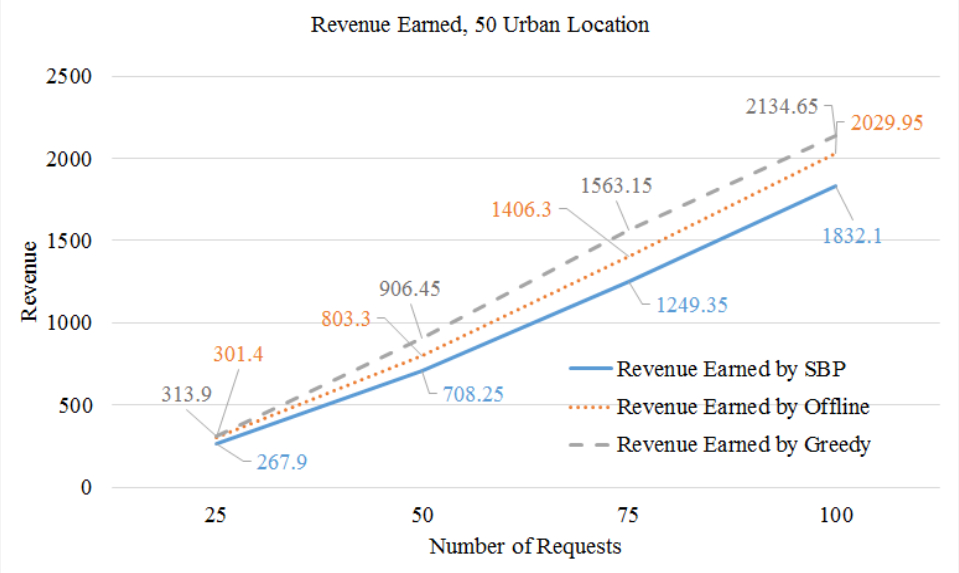

Figure 3 Revenue earned for urban setting.

urban setting. The graphs show that online-SBP improves relative to offline-SBP as the ratio between the average edge weight and the maximum edge weight (T/f) increases. These ratios are 58%, 61%, and 63% for the rural, suburban, and urban settings, respectively. When this ratio is relatively low the offline algorithm is more likely to be able to serve multiple requests within a single time segment and can better take advantage of having all the requests available. Specifically for 25[1] requests, offline serves on average 1.8, 1.2, and 1.2 requests per time segment for rural, suburban, and urban, respectively. Since SBP is unaware of all the future requests, it is always less likely than offline to serve multiple requests within a time segment. Specifically, SBP serves on average 1.6, 1.2, and 1 requests per time segment, respectively. These values show that as the number of requests served per time unit decreases for offline, the two algorithms' performances become comparable, but the performance is most closely matched in the suburban setting when the number of requests served per time unit is the same for both online and offline-SBP.

While the greedy algorithm often slightly outperforms SBP, it is important to note that the greedy algorithm has an unbounded competitive ratio, while SBP has both a constant-competitive worst-case guarantee (as proven in Section 4) and it also performs on par with a basic greedy algorithm in these "average-case" experimental simulations.

We also determined how our algorithm compares to the overall total revenue released (i.e. if there are no server capacity or time constraints) and the percentages are as follows. In the rural setting, SBP earns 34%, 21%, 19%, and 14% of the total revenue for 25, 50, 75, and 100 requests respectively. In the suburban and urban settings, these values are, respectively, 42%, 27%, 21%, and 17% (for suburban) and 81%, 55%, 43% and 36% (for urban). For more extensive experimental results, including non-uniform distributions, please refer to the full version of the paper.

[1] Similar trends occur for 50, 75, and 100 requests.

6 Conclusions

In this work we present an initial study on the On-Line Dial-a-Ride Problem with revenues (ROLDARP) in weighted graphs. We show that the competitive ratio is unbounded for any deterministic online algorithm for the problem, unless we assume edge weights are bounded and discount the revenue earned by OPT in the last time segment. We provide an algorithm, SBP, that is 6-competitive with OPT, minus the revenue OPT earns in the last two time segments. We also provide a lower bound of 4 on the competitive ratio of SBP. Hence, there currently remains a gap between the upper and lower bounds on the competitive ratio of SBP. Other remaining open questions include: whether there are other more efficient and/or more competitive algorithms for ROLDARP than what we have proposed, both in the uniform metric as well as for weighted graphs, and whether ROLDARP in the uniform metric is NP-hard.

Acknowledgements. The authors would like to thank the anonymous reviewers for their very helpful comments.

References

1 Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 639–650. Springer, 2000.

2 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman 1. *Algorithmica*, 29(4):560–581, 2001.

3 Nabila Azi, Michel Gendreau, and Jean-Yves Potvin. A dynamic vehicle routing problem with multiple delivery routes. *Annals of Operations Research*, 199(1):103–112, 2012.

4 Ananya Christman and William Forcier. Maximizing revenues for on-line dial-a-ride. In *International Conference on Combinatorial Optimization and Applications*, pages 522–534. Springer, 2014.

5 Minnesota City of Plymouth. Plymouth metrolink dial-a-ride. URL: `http://www.plymouthmn.gov/departments/administrative-services-/transit/plymouth-metrolink-dial-a-ride`.

6 Stagecoach Corporation. Dial-a-ride. URL: `http://stagecoach-rides.org/dial-a-ride/`.

7 Metropolitan Council. Transit link: Dial-a-ride small bus service. URL: `https://metrocouncil.org/Transportation/Services/Transit-Link.aspx`.

8 Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

9 Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

10 Patrick Jaillet and Michael R Wagner. Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses. *Operations Research*, 56(3):745–757, 2008.

11 Patrick Jaillet and Michael R. Wagner. Online vehicle routing problems: A survey. *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 221–237, 2008.

12 Yannick Kergosien, Ch. Lente, D. Piton, and J.-C. Billaut. A tabu search heuristic for the dynamic transportation of patients between care units. *European Journal of Operational Research*, 214(2):442–452, 2011.

13 Sven O. Krumke, Willem E. de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On minimizing the maximum flow time in the

online dial-a-ride problem. In *International Workshop on Approximation and Online Algorithms*, pages 258–269. Springer, 2005.

14 Sandro Lorini, Jean-Yves Potvin, and Nicolas Zufferey. Online vehicle routing and scheduling with dynamic travel times. *Computers & Operations Research*, 38(7):1086–1090, 2011.

15 Michael Schilde, Karl F. Doerner, and Richard F. Hartl. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers & Operations Research*, 38(12):1719–1730, 2011.