On-Line Algorithms for the Dynamic Traveling Repair Problem

Sandy Irani* Xiangwen Lu † Amelia Regan ‡

August 7, 2002

Abstract

We consider the dynamic traveling repair problem (DTRP) in which requests with deadlines arrive through time on points in a metric space. Servers move from point to point at constant speed. The goal is to plan the motion of servers so that the maximum number of requests are met by their deadline. We consider a restricted version of the problem in which there is a single server and the length of time between the arrival of a request and its deadline is constant. We give upper bounds for the competitive ratio of two very natural algorithms as well as several lower bounds for any deterministic algorithm. Most of the results in this paper are expressed as a function of β, the diameter of the metric space. In particular, we prove that the upper bound given for one of the two algorithms is within a constant factor of the best possible competitive ratio.

1 Introduction

In the dynamic traveling repair problem (DTRP), servers move from point to point in a metric space. The speed of each server is constant, so the time it takes to travel from one point to another is proportional to the distance between the two points. Time is continuous and at any moment a request for service can arrive at any point in the space. Each job also specifies a deadline. If a job is to be serviced, a server must reach the point where the request originated by its deadline. The goal of the algorithm is to service as many incoming requests as possible by their deadlines. We consider *online* algorithms which learn about a new request only when it arrives in the system. The DTRP has many practical applications. Examples include routing utility and other repair fleets as well as routing of tanker trucks [6, 7].

In this paper, we consider the following restrictions to the DTRP problem:

- **Single Server:** We address the problem in which there is only one server.
- **Window sizes are uniform:** We restrict our attention to the case where the time between a job's arrival and its deadline (the *window* of a request) is a fixed period W for all jobs. For the remainder of this paper, we will normalize all values so that the window length is 1.

*Information and Computer Science Department, UC Irvine, 92697, irani@ics.uci.edu. Supported in part by NSF grants CCR-0105498 and CCR-9625844 and by ONR Award N00014-00-1-0617.

†Institute of Transportation Studies, UC Irvine, 92697, xlu@uci.edu. Partially supported by NSF Grant 9875675.

‡Department of Civil and Environmental Engineering, UC Irvine, 92697, aregan@uci.edu. Partially supported by NSF Grant 9875675.

- **Notification time:** One can consider variations of this problem depending on when the online algorithm is required to commit to accepting or rejecting a particular request. We adopt this as a parameter of the problem. Thus, an online algorithm must decide whether to accept or reject a request within time α of its arrival. We call α the *notification deadline*. If $\alpha = 0$, the server must decide immediately upon the arrival of a job into the system whether it will be served. If $\alpha = 1$, the decision is made only when the request is served or its deadline has passed.

  Most of the results in this paper focus on the case where $\alpha = 1$, although there is one lower bound which pertains to the general case. The situation in which the notification deadline is 1 is in fact reasonable operationally. Many cable TV or other repair fleets accept all requests and sub-contract, sometimes at the last minute, requests they are unable to service with their primary fleet. Local truckload trucking operations, which can be modeled as a DTRP with asymmetric travel costs, typically operate in the same way. A primary carrier accepts all requests for service and then sub-contracts to local owner operators if it is unable to service requests within their time windows.

- **Bounded diameter:** As we shall see in the results below, it is necessary to consider metric spaces of bounded diameter. We assume that the diameter of the metric space is bounded by a constant β. Our results are expressed as a function of β. In order to have a competitive algorithm, it is required that β be less than 1.

- **Service time:** In most of our results, we assume that once the server arrives at the location of the request, it is served instantly. A natural extension of this problem would be to have a service time required for each request. We extend the upper bound of each of our algorithms to apply to the case where each request has a fixed service time s.

A word about metric spaces: some previous work on related problems ([5]) focuses on continuous metric spaces in which the path between any two points is continuous, formed by points in the metric space. In particular, this implies that the server can do a U-turn at any point in time. Our results hold for the more general case where the path between two points may not be comprised of points in the metric space and the server, once embarked on an edge between two points, must traverse the edge before changing directions. In this case, β indicates the longest time that it takes the server to reach any point in the metric space maximized over all possibilities for its current location and destination.

We use competitive analysis to evaluate online algorithms. This has now become the most widely accepted means of evaluating an online algorithm. In competitive analysis, the performance of an online algorithm is compared to the performance of the optimal offline algorithm which knows about all future jobs. In the problem addressed in this paper, the algorithms are seeking to maximize their *benefit* for a particular sequence of request arrivals which is the number of requests satisfied by their deadline. Let $B_A(\sigma)$ denote the benefit obtained by algorithm A on request sequence σ. Let OPT denote the optimal offline algorithm. The algorithm A is said to be c-competitive if there is a constant d such that for every sequence σ,

$$c \cdot B_A(\sigma) + d \geq B_{OPT}(\sigma).$$

The *competitive ratio* of the algorithm A, denoted c_A, is the infimum over c such that A is c-competitive.

2 Related Work

The dynamic traveling repair problem (DTRP) falls into a more general class of problems known as stochastic vehicle routing. This class includes the dynamic traveling salesman problem (DTSP), the probabilistic traveling salesman problem (PTSP), probabilistic traveling salesman location problem and other problems such as the probabilistic shortest path problem and the dynamic vehicle allocation problem. Powell, Jaillet and Odoni provide an excellent review of these problems [15]. In the DTSP, customer locations are known in advance and service requests arrive according to a Poisson point process at each node. The objective is to determine a dispatching strategy which minimizes customers' expected waiting time. This problem was introduced by Psarafits [16]. In the PTSP, an a priori tour must be constructed for a network in which each node has a given probability of requiring a visit [12]. In the DTRP it is typically assumed that the input is generated by a known probability distribution. In this paper, we use competitive analysis to analyze algorithms for the DTRP in which the input sequence can be arbitrary.

From the online algorithms literature, this problem most closely resembles the On-Line Traveling Salesman Problem (OLTSP) studied by Ausiello *et al.*[5]. Each job has a release time and can not be serviced before this time. Requests do not have deadlines. Instead, all jobs must be serviced and the goal is to minimize the total distance traveled by the server in serving all the requests. They give a 2.5-competitive algorithm for all *continuous* metric spaces in which the server can turn around at any point in time and return to the last point visited. They also consider a variant of the problem called the *Homing*-OLTSP in which the server is required to return to its departure point before serving another request. The offline version of OLTSP is called Vehicle Scheduling or Vehicle Routing. Augustine and Seiden examine this problem where each job also has a handling time during which the server must be located at the point of request to service the job [3]. Similar to OLTSP is the Online Traveling Repair Problem (OLTRP) which is the same as OLTSP except that the quantity to be minimized is the weighted sum of the completion times of all the requests [13]. The *completion time* of a request is defined to be the time traveled on the tour until the request is served.

Another related problem is the dial-a-ride problem in which requests arrive in a metric space. Each request consists of a pick-up point and a drop-off point. To satisfy a request, an object must be picked up from the pick-up point and dropped off at the drop-off point. The cost of an algorithm is some function of the arrival times of the requests and the route chosen by the server. Variations include minimizing the maximum or average flow time [10], minimizing the maximum completion time of any request [2, 9, 11] or minimizing the sum of the completion times over all requests [9, 13]. Other variations of the problem include different server capacities and precedence constraints on the requests [13]. The primary difference between all of these problems (OLTSP, OLTRP, dial-a-ride) and the problem we study here is the absence of deadlines and the requirement that all requests be satisfied. The problem addressed in this paper has the problem of admission control (i.e. which requests to serve) as well as how to serve the requests that have been accepted by their deadlines.

Other related work includes online graph exploration problems in which all nodes of a graph must be visited and the edges of the graph are revealed only when the server has visited an incident node [8]. In our problem, the entire graph (or metric space) is known in advance and the locations of requests are revealed in an on-line manner. Another related problem is the k-server problem [14]. In the k-server problem, it is assumed that servers can travel instantaneously to their destination, and the goal is to minimize the total distance traveled by all servers. Also, in the k-server problem, requests must be visited in the order in which they arrive but in the problem addressed here, the server can choose the order in which to satisfy the requests. In the k-client problem [1], each of k-clients has at most one active request on a point in the metric space. A single server must serve all requests. When a request belonging to a client is served, that client may choose to introduce another request. As in the k-server problem (and unlike our problem), the goal is to minimize the total distance traveled. Another difference between the k-client problem and the problem that we address in this paper is that in the k-client problem, the order in which the requests arrive depends on the behavior of the algorithm.

3 Summary of Results

We present two very natural algorithms for the DTRP and prove upper bounds for their competitive ratios. All of our results are given as a function of β which is the maximum time required to travel between the two farthest points in the metric space. The particular value of β for a metric space determines which bound is stronger.

3.1 The BATCH Algorithm

The first algorithm is called the BATCH algorithm. Time is divided into intervals of length 1/2. At the beginning of an interval, the jobs which arrived in the previous interval are considered. The algorithm then determines the maximum number of requests from this set that it can reach from its current position in the next interval of 1/2 and follows that path. Thus, the set of requests considered in a given interval of 1/2 are those requests which arrived in the previous interval and expire in the next interval. The algorithm must solve an instance of the prize-collecting TSP problem with a fixed starting position and an arbitrary ending location for every interval. (See [4] for a discussion of the prize-collecting TSP). This means that BATCH is not a polynomial time algorithm. One possibility is that the number of requests that arrive in a single interval will be small enough that the prize-collecting TSP can be solved exactly. Another possibility is that an approximation algorithm can be used in which case the approximation factor would be multiplied by the competitive ratio. Note that BATCH requires that the notification deadline (α) be at least 1/2. We prove that BATCH is $(3/(1-2\beta))$-competitive. We then give a lower bound for BATCH which shows that our analysis is tight. This means that BATCH is not competitive for $\beta \geq 1/2$. We also show that in the case where each request requires a fixed service time s, BATCH is $(3/(1-2\beta-2s))$-competitive on any metric space with diameter β.

3.2 The Double-Gain Algorithm

An appealing feature of Double-Gain (DG) is that it is well defined for the more general version of DTRP with non-uniform windows. Our proof for the competitiveness of DG, however, only applies for uniform window lengths.

The state of an algorithm is defined by its current location and the set of outstanding requests which have not yet expired. The maximum number of outstanding requests which can be served given that the algorithm is in state S at time t is well defined. The route which achieves this maximum will be called the *new* route for time t. If more than one route achieves the maximum, an optimal route can be chosen arbitrarily. $N(t)$ will denote the set of requests which are served on the new route for time t. At any given moment in which there are outstanding requests in the system, the algorithm has a route that it is currently executing which we will call the *current* route. If the algorithm finishes its current route at time t, it begins immediately executing the new route for time t. If there are no outstanding requests in the system which the algorithm can reach, the current route for the algorithm is simply to stay put. Let $C(t)$ denote the set of outstanding requests which have yet to be served on the algorithm's current route at time t. At any given point in time t, the server will switch to the new route at time t if the following condition holds:

$$|N(t) - C(t)| \geq 2(|C(t) - N(t)|).$$

That is, the algorithm switches routes if the number of requests gained is at least twice the number of requests lost by the switch. Note that the proof requires that the new route must be optimal. The proof does not hold if the algorithm simply picks any route for which the number of requests gained is at least twice the number of requests dropped.

We prove that the competitive ratio of Double-Gain is at most

$$4\left\lceil \frac{2}{1-\beta} \right\rceil \left(\left\lceil \frac{2}{1-\beta} \right\rceil + 1 \right) + 1.$$

If each request requires service time s, the bound becomes

$$4\left\lceil \frac{2}{1-\beta-s} \right\rceil \left(\left\lceil \frac{2}{1-\beta-s} \right\rceil + 1 \right) + 1.$$

Note that Double-Gain is not a polynomial time algorithm since it requires solving for $N(t)$ every time a new request arrives. If an r-approximation is used to calculate $N(t)$ the bound for the competitive ratio becomes

$$4r\left\lceil \frac{2}{1-\beta} \right\rceil \left(\left\lceil \frac{2}{1-\beta} \right\rceil + 1 \right) + 1.$$

3.3 Lower Bounds

We first show that for any notification deadline α, and any $\beta > \alpha$, there is a metric space with diameter β for which no competitive algorithm exists. We then prove two lower bounds which hold for any deterministic online algorithm and any value for α. Each lower bound is a function of β. Thus, the value of β determines which lower bound is strongest. We prove that there is no deterministic algorithm which will achieve a competitive ratio lower than the following bound:

$$\max\left\{2, \frac{\beta}{(1-\beta)^2}\right\}.$$

A different metric space is required for each of the two lower bounds. The proof for the last bound of $\beta/(1-\beta)^2$ holds only when $\beta > 1/2$. However, if $\beta \leq 1/2$, the lower bound of 2 will be stronger anyway. Note that this bound shows that the upper bound proven for Double-Gain is within a constant factor of the best possible competitive ratio.

4 Bounds for the BATCH Algorithm

Theorem 1 *For any $\alpha > 1/2$ and any metric space with diameter $\beta < 1/2$, BATCH is $3/(1-2\beta)$-competitive.*

Proof. Consider a sequence of arrivals σ and the schedule which OPT chooses for σ. Time is divided into intervals of length $1/2$. Number the intervals from 1 to m. Interval m is the last interval in σ in which any jobs arrive. Let $S_{i,j}$ be the set of jobs which arrive during interval i and which OPT schedules during interval j. Let $s_{i,j} = |S_{i,j}|$. Clearly, if $i < j-2$ or $i > j$, then $S_{i,j} = \emptyset$. Recalling that $B_{OPT}(\sigma)$ represents the benefit obtained by OPT on request sequence σ,

$$B_{OPT}(\sigma) = \sum_{j=1}^{m+2} (s_{j-2,j} + s_{j-1,j} + s_{j,j}).$$

Suppose that BATCH is at location q at the beginning of interval i. It will consider the set of requests S_{i-1} which arrived during interval $i-1$ and find the tour which reaches the most requests from S_{i-1}, starting at q and lasting for at most $1/2$. Note that $S_{i-1,i-1} \cup S_{i-1,i} \cup S_{i-1,i+1} \subseteq S_{i-1}$. Also note that the optimal can serve each of $S_{i-1,i-1}$, $S_{i-1,i}$ and $S_{i-1,i+1}$ during a single interval. This means that there is a tour of length at most $1/2$ which covers all of $S_{i-1,i-1}$, given that the starting position can be chosen arbitrarily. The same is true for $S_{i-1,i}$ and $S_{i-1,i+1}$.

Now since β is the longest time it takes to travel between two points, BATCH can pick its optimal starting location, take time β to travel to that location and has a remaining time of $1/2-\beta$ to service requests. Thus, one of the options is to pick the largest of $S_{i-1,i-1}$, $S_{i-1,i}$ and $S_{i-1,i+1}$. Call it $S_{i-1,j}$. Consider the tour which covers all the requests in $S_{i-1,j}$ in time $1/2$. Now pick the portion of the path which is $(1/2-\beta)$ long and covers the most requests. Since the length of the entire path is $1/2$, this portion of the path will reach at least $(1-2\beta)s_{i-1,j}$ requests. BATCH has the option of traveling to the beginning of that portion of the path and visiting all the requests along that portion of the path. This means that the number of jobs which BATCH visits during interval i is at least

$$(1-2\beta)\max\{s_{i-1,i-1}, s_{i-1,i}, s_{i-1,i+1}\}$$

and

$$\begin{aligned} & B_{BATCH}(\sigma) \\ \geq \; & \sum_{i=2}^{m+1} (1-2\beta)\max\{s_{i-1,i-1}, s_{i-1,i}, s_{i-1,i+1}\} \end{aligned}$$

$$\begin{aligned}
&\geq \left(\frac{1-2\beta}{3}\right)\sum_{i=2}^{m+1}(s_{i-1,i-1}+s_{i-1,i}+s_{i-1,i+1}) \\
&= \left(\frac{1-2\beta}{3}\right)\sum_{j=1}^{m+2}(s_{j-2,j}+s_{j-1,j}+s_{j,j}) \\
&= \left(\frac{1-2\beta}{3}\right) B_{OPT}(\sigma).
\end{aligned}$$

□

The following corollary generalizes the upper bound for BATCH to apply in cases in which each request requires a service time of s. We assume that each request must be completed by its deadline.

Corollary 2 *For $\alpha > 1/2$ and any $\beta + s < 1/2$, if each request requires service time s, then on any metric space with diameter β, BATCH is $3/(1-2(\beta+s))$-competitive.*

Proof. The proof of Theorem 1 goes through with a few changes. $S_{i,j}$ is the number of jobs which arrive during interval i and which OPT completes in interval j. It is no longer true that there is a tour of length $1/2$ which can satisfy any $S_{i,j}$ since OPT could be servicing a request at the beginning of the interval. Instead, we say that there is a tour of length $1/2$ which can service all the requests in $S_{i,j}$ except for the first one which may be only reached but not serviced.

For interval i, BATCH can pick the largest of $S_{i-1,i-1}$, $S_{i-1,i}$ and $S_{i-1,i+1}$. Call it $S_{i-1,j}$. Pick the portion of the path which is $1/2-\beta-s$ long and covers the most points in time in which OPT finishes a request. This will be at least $s_{i-1,j}(1-2\beta-2s)$ requests. BATCH can travel a distance β to the first request in the path, service the first request in time s and then service all the remaining requests in that portion of the path. □

The following result shows that our analysis for BATCH is tight.

Theorem 3 *For any $\beta < 1/2$, there is a metric space $\mathcal{M}$ with diameter β such that for any d and any constant $c < 3/(1-2\beta)$, there is a sequence of arrivals σ such that*

$$c \cdot B_{BATCH}(\sigma) + d < B_{OPT}(\sigma).$$

Proof. Pick a small constant δ such that $1/2\delta$ is an integer m. δ can be chosen arbitrarily small. Let $r = \lceil (1-2\beta)/2\delta \rceil = \lceil (1/2-\beta)/\delta \rceil$. Now we position $3m$ points in the metric space, each a distance of δ away from each other. Call this set A. There will be another set of points B which will consist of $r+1$ points which are also each a distance of δ away from each other. Each point in A is a distance of β away from any point in B.

The points in A will be labeled $a_{i,j}$, where i is an integer in the range $0, 1, \ldots, m-1$ and $j \in \{0, 1, 2\}$ The points in B will be labeled $b_0, b_1, \ldots, b_r$.

No jobs arrive on points in A during the first interval. Let ϵ be a small constant such that $\epsilon << \delta$. Requests will arrive during interval $b \geq 2$ on points in A as follows:

For b MOD $3 \equiv 0$: For each $i \in \{0, 1, \ldots, m-1\}$:

- A request arrives at $a_{i,0}$ at time $b/2 + i\delta - \epsilon$.

- A request arrives at $a_{i,1}$ at time $b/2 + i\delta$.
- A request arrives at $a_{i,2}$ at time $b/2 + i\delta + \epsilon$.

For b MOD $3 \equiv 1$: For each $i \in \{0, 1, \ldots, m-1\}$:

- A request arrives at $a_{i,0}$ at time $b/2 + i\delta + \epsilon$.
- A request arrives at $a_{i,1}$ at time $b/2 + i\delta - \epsilon$.
- A request arrives at $a_{i,2}$ at time $b/2 + i\delta$.

For b MOD $3 \equiv 2$: For each $i \in \{0, 1, \ldots, m-1\}$:

- A request arrives at $a_{i,0}$ at time $b/2 + i\delta$.
- A request arrives at $a_{i,1}$ at time $b/2 + i\delta + \epsilon$.
- A request arrives at $a_{i,2}$ at time $b/2 + i\delta - \epsilon$.

In addition, in the middle of every interval, a request will arrive on each point in B.

This sequence can continue for an arbitrary number of intervals, say T. How does BATCH service the requests? During the second interval, it tries to service some of the requests that arrived during the first interval. Since these are all on points in B, BATCH will move to a point in B and in the best case, service all the $r+1$ requests which arrived. We will show that in each subsequent interval, BATCH will remain on points in B and will get $r+1$ requests per interval. Note that in each interval, exactly one request arrives on each point in A and B. If BATCH does not travel to set A, then it can get all $r+1$ requests which arrive in B. If it does travel to A, then it will spend β time in transit from B to A. That means it has a remaining $1/2 - \beta$ time to go from point to point (either in A or B). Since each point is δ away from every other point, it can get to at most $\lceil (1/2 - \beta)/\delta \rceil = r$ requests which arrive in the previous interval. Note that although it could in principle serve more requests, we only consider the version of BATCH which restricts itself to serving requests which arrive in the previous interval. Since the first option yields more requests satisfied, BATCH will stay in B and service the $r+1$ requests. This amounts to a total of $T(r+1)$ requests satisfied over the course of the entire sequence.

Now we will show that the optimal algorithm can satisfy a total of $3m(T-2)$ requests over the entire sequence. The optimal will forgo any requests that arrived in the first interval. Then for interval $b \geq 2$, it will visit each node $a_{0,j}, a_{1,j}, \ldots, a_{m-1,j}$ in turn where $j = b$ MOD 3. It arrives at node $a_{i,j}$ at time $b/2 + i\delta$ where there are three requests waiting: one that arrived at time $(1/2)(b-2) + i\delta + \epsilon$ (and expires at $b/2 + i\delta + \epsilon$), one that arrived at time $(1/2)(b-1) + i\delta$ (and expires at time $(1/2)(b+1) + i\delta$) and one that arrived at time $b/2 + i\delta - \epsilon$. Thus, after the third interval, the optimal algorithm will get $3m$ requests per interval. It will get $2m$ in interval 3 and m in interval 2. Thus, the total number of requests satisfied will be $3m(T-2)$. Now pick constants d and $c < 3/(1-2\beta)$. Say $c = [3/(1-2\beta)] - \gamma$. We will show that T can be picked large enough and δ small enough so that

$$\begin{aligned}
c \cdot B_{BATCH}(\sigma) + d &\leq B_{OPT}(\sigma) \\
\left[\frac{3}{1-2\beta} - \gamma\right] T(r+1) + d &\leq 3m(T-2)
\end{aligned}$$

$$\left[\frac{3}{1-2\beta}-\gamma\right]+\frac{d}{T\left(\left\lceil\frac{1-2\beta}{2\delta}\right\rceil+1\right)} \quad \leq \quad \frac{\frac{3}{2\delta}(T-2)}{T\left(\left\lceil\frac{1-2\beta}{2\delta}\right\rceil+1\right)}$$

This holds if

$$\left[\frac{3}{1-2\beta}-\gamma\right]+\frac{d}{T\left(\frac{1-2\beta}{2\delta}+1\right)} \leq \frac{\frac{3}{2\delta}(T-2)}{T\left(\frac{1-2\beta}{2\delta}+2\right)}$$

Which is the same as

$$\left[\frac{3}{1-2\beta}-\gamma\right]+\frac{d}{T\left(\frac{1-2\beta}{2\delta}+1\right)} \leq \left(\frac{3\left(1-\frac{2}{T}\right)}{1-2\beta+4\delta}\right)$$

As T grows and δ shrinks, the right hand side gets arbitrarily close to $3/(1-2\beta)$ and the left hand side gets arbitrarily close to $[3/(1-2\beta)]-\gamma$, so the inequality will hold. □

5 An Upper Bound for Double-Gain

Theorem 4 *For $\alpha = 1$ and any metric space with diameter $\beta < 1$, the competitive ratio of Double-Gain is at most*

$$4\left\lceil\frac{2}{1-\beta}\right\rceil\left(\left\lceil\frac{2}{1-\beta}\right\rceil+1\right)+1.$$

Proof. We will consider a slightly more general algorithm in which the algorithm changes its current route to be the new route at time t if

$$|N(t)-C(t)| \geq c(|C(t)-N(t)|),$$

for some $c > 1$. Then we will show that 2 is the optimal choice for c. Allowing $c = 2$ achieves our bound.

Fix an input sequence σ. Partition time into periods of length $\phi = (1-\beta)/2$. The i^{th} period ends at time ϕi. At the end of each period, we will consider the number of outstanding requests in the Double-Gain's current route. Recall that at time t, the set of outstanding requests in the current route is denoted by $C(t)$. We will prove two facts.

1. $\frac{c}{c-1}\left\lceil\frac{1}{\phi}\right\rceil B_{DG}(\sigma) \geq \sum_i |C(\phi i)|$.
2. $B_{OPT}(\sigma) \leq c\left\lceil\frac{1+\phi}{\phi}\right\rceil \sum_i |C(\phi i)| + B_{DG}(\sigma)$.

The competitive ratio is then

$$\frac{c^2}{c-1}\left\lceil\frac{2}{1-\beta}\right\rceil\left(\left\lceil\frac{2}{1-\beta}\right\rceil+1\right)+1.$$

The expression is minimized for $c = 2$ which gives Theorem 4.

We start by proving the first fact. For each request that ever appears in Double-Gain's current route, consider a continuous interval of time in which it is in the current route. (Some requests may appear in the algorithm's current route, drop out and then reappear in the algorithm's current route. This is treated as two separate intervals.) If these intervals are depicted on a horizontal time axis, the left endpoint of an interval is its *beginning* endpoint and its right endpoint is its *final* endpoint. $|C(t)|$ is the number of intervals that contain point t. Since every request expires or is served within time 1 of its arrival, each interval can cover at most $\lceil 1/\phi \rceil$ period endings. This means that

$$\sum_i |C(\phi i)| \leq \lceil 1/\phi \rceil I, \tag{1}$$

where I is the total number of intervals.

An interval ends either because the request was satisfied or because the algorithm chose a new route and dropped the request from its current route. We call the former *satisfied* intervals and the latter *unsatisfied* intervals. Consider a point in time when the algorithm chooses a new route. Suppose that x unsatisfied intervals end at this moment. Then it must be the case that at least cx intervals begin at this moment as well. (This is the requirement for choosing a new route).

Now we will assign a weight to each interval. We start with each interval having a weight of one. We will move a pointer from left to right along the time axis, starting with the earliest beginning endpoint and ending with the latest final endpoint. The location of the pointer represents a point in time. As the pointer moves, we will maintain two properties. The first is that the weight of each interval is at most

$$\sum_{i=0}^{\infty} \left(\frac{1}{c}\right)^i = \frac{c}{c-1}.$$

The second is that any interval whose beginning endpoint occurs later than (or to the right of) the current position of the pointer has a weight of 1.

Suppose that at time t the pointer reaches the beginning endpoint of an unsatisfied interval. Let x be the number of unsatisfied intervals whose final endpoint is at time t. By the conditions of the algorithm, there are at least $\lceil cx \rceil$ intervals whose beginning endpoint is at time t. We know that the weight associated with each of these $\lceil cx \rceil$ intervals is 1 since until now their beginning endpoints have been to the right of the pointer. Let W be the total weight of all the unsatisfied intervals who have their final endpoint at time t. By induction, we know that $W \leq cx/(c-1)$. Set the weight of all these intervals to 0. Select $\lceil cx \rceil$ intervals whose beginning endpoint is at time t and add $W/\lceil cx \rceil$ to the weight of each of these intervals. Thus, the new weight of the selected $\lceil cx \rceil$ intervals is at most

$$1 + \frac{1}{\lceil cx \rceil}\frac{cx}{c-1} \leq \frac{c}{c-1}.$$

This ensures that property one is maintained. No interval whose beginning endpoint is later than the current location of the pointer has changed weight, so the second property still holds as well.

At the beginning of this process, the total weight is equal to the number of intervals. Furthermore, throughout the process, the total weight of all the intervals does not change. At the end of the process, the weight of each unsatisfied interval is 0 and the weight of any interval is at most $c/(c-1)$. This means that the number of satisfied intervals is at least $I(c-1)/c$. Since $B_{DG}(\sigma)$ is the number of requests served by Double-Gain,

$$\frac{c}{c-1} B_{DG}(\sigma) \geq I.$$

Putting this together with Inequality (1), implies the Fact 1.

Now we will establish Fact 2. We will call a request an *optimal-only* request if it is served by the optimal algorithm but not by Double-Gain. We will show that the number of optimal-only requests is at most

$$c \left\lceil \frac{1+\phi}{\phi} \right\rceil \sum_i |C(\phi i)|.$$

We do this by bounding the number of requests which the optimal can serve that arrive in a period of length ϕ. Suppose the period starts at time $t-\phi$ and lasts through time t. In determining $N(t)$, one possible option for Double-Gain is to consider only the jobs which arrive in the interval from $t-\phi$ to time t which it has not yet served, spend time β moving to the optimal starting location and spend time ϕ serving as many requests that can be served in an interval of length ϕ. Note that $2\phi+\beta=1$, so none of the requests which arrived in the interval from $t-\phi$ to time t will have expired. Let N be the set of jobs that would be satisfied by this course of action. Note that $|N(t)| \geq |N|$. If Double-Gain does not change routes at time t, then $c|C(t)-N(t)| > |N(t)-C(t)|$ which implies that $c|C(t)| \geq |N(t)| \geq |N|$. If Double-Gain does change routes at time t, then the new route for time t becomes the current route for time t. This means that $|C(t)|$ is the maximum number of future requests that Double-Gain can satisfy among all outstanding requests, given its current location. Since N is the set of future requests which can be satisfied using a specific route, $|C(t)| \geq |N|$. In either case, $c|C(t)|$ is at least the number of requests gained by the optimal algorithm which is allowed *(1)* to choose any set of requests which arrive in the interval from $t-\phi$ to time t and which have not been served by Double-Gain by time t, *(2)* relocate to an optimal location and *(3)* serve as many of these requests which can be served in an interval of length ϕ. By further restricting the set of possible requests to be requests that Double-Gain never serves (i.e. optimal-only requests), this quantity can only decrease.

Any job which arrives during the interval $[t-\phi, t)$ must be scheduled in the interval $[t-\phi, t+1)$. We have established that during any subinterval of $[t-\phi, t+1)$ of length ϕ, the optimal algorithm can serve at most $c|C(t)|$ of the requests which arrive during $[t-\phi, t)$ and which Double-Gain does not serve. This means that at most $c|C(t)|\lceil(1+\phi)/\phi\rceil$ requests which arrive during the interval $[t-\phi, t)$ and which Double-Gain does not serve can be scheduled by the optimal algorithm at any time. Using ϕi for t, we get that the number of optimal-only requests which arrive in period i is at most

$$c|C(\phi i)| \left\lceil \frac{1+\phi}{\phi} \right\rceil.$$

This means that the total number of optimal-only requests is at most

$$c \left\lceil \frac{1+\phi}{\phi} \right\rceil \sum_i |C(\phi i)|.$$

This establishes Fact 2. □

Corollary 5 *Suppose that an r-approximation algorithm is used to calculate $N(t)$ instead of an exact* *algorithm. Then the competitive ratio of Double-Gain would be*

$$4r\left\lceil\frac{2}{1-\beta}\right\rceil\left(\left\lceil\frac{2}{1-\beta}\right\rceil+1\right)+1.$$

Proof. The main thing that changes in the proof of Theorem 4 is that $r|N(t)| \geq |N|$. The factor of r carries through so that Fact 2 becomes

$$B_{OPT}(\sigma) \leq cr\left\lceil\frac{1+\phi}{\phi}\right\rceil\sum_i |C(\phi i)| + B_{DG}(\sigma).$$

□

The next bound shows how the competitive ratio changes with a fixed service time.

Corollary 6 *For $\alpha = 1$, if each requires service time s and $\beta + s < 1$, then for any metric space with diameter β, the competitive ratio of Double-Gain is at most*

$$4\left\lceil\frac{2}{1-\beta-s}\right\rceil\left(\left\lceil\frac{2}{1-\beta-s}\right\rceil+1\right)+1.$$

Proof. The proof follows the proof of Theorem 4 very closely. Let $\phi = (1-\beta-s)/2$. The proof of Fact 1 remains the same.

For the proof of Fact 2, in determining $N(t)$ we consider the possiblity of spending time β moving to an optimal starting location and spending time $\phi+s$ serving requests. Note that now $2\phi+\beta+s = 1$. Let N be the set of jobs which would be satisfied by this course of action. The reasoning is the same that $c|C(t)|$ is at least the number of requests gained by the optimal algorithm which is allowed *(1)* to choose any set of requests which arrive in the interval from $t-\phi$ to time t and which have not been served by Double-Gain by time t, *(2)* relocate to an optimal location and *(3)* serve as many of these requests which can be served in an interval of length $\phi + s$. Note that it is always in the optimal algorithm's best interest to move to a location where a request will be served and start the interval by serving a request. This means that any interval of length ϕ, can contain at most $c|C(t)|$ points in time in which the optimal finishes serving a request which arrived in the interval from $t-\phi$ to time t and was not served by Double-Gain. Thus, the optimal algorithm can serve at most $c|C(t)|\lceil(1+\phi)/\phi\rceil$ optimal-only requests which arrived during the interval from $t-\phi$ to time t. The rest of the proof holds unchanged.

□

6 General Lower Bounds

In all of the lower bounds presented in this section, the algorithm is allowed to do a U-turn along any edge in the metric space. The following theorem shows that α (the notification deadline) must be at least β in order for a competitive algorithm to exist.

Theorem 7 *If $\alpha < \beta$, then there is a metric space on which no deterministic online algorithm can achieve a bounded competitive ratio.*

Proof. Let c be an arbitrary constant. The metric space will consist of many points, all β away from each other. At time $i \cdot \alpha$, c^i requests will be requested at a new unused point in the metric space. This starts with $i = 0$ and ends either when the algorithm decides not to visit a requested point or when $i > (1-\beta)/(\beta-\alpha)$. Let's say that the last group arrives at time $k\alpha$.

Consider two cases:

$k \leq (1-\beta)/(\beta-\alpha)$: In this case, the adversary will serve all the requests. Note that this is possible since the server will arrive at the i^{th} point at time $\beta(i+1)$ and the requests arrived at time αi. Since $k \leq (1-\beta)/(\beta-\alpha)$, we know that $\beta(k+1) \leq \alpha k + 1$ and the requests will not have expired. Thus, the optimal serves $1 + c + c^2 + \cdots c^k = (c^{k+1}-1)/(c-1)$ requests. The online algorithm only serves $1 + c + c^2 + \cdots c^{k-1} = (c^k - 1)/(c-1)$ requests since the online algorithm decided not to serve the last set. This gives a ratio of $(c^{k+1}-1)/(c^k-1) \geq c - 1$.

$k > (1-\beta)/(\beta-\alpha)$: In this case, the online algorithm can not make it to the last set. This is because the algorithm must decide whether or not to satisfy each set of requests before the next set arrives. By the time the last set has arrived at time αk, it has committed to travel a total of βk. If it commits to the last set, it will have to travel $\beta(k+1)$, which it can not do by time $1 + \alpha k$ since $k > (1-\beta)/(\beta-\alpha)$. The adversary will skip the first one and get the rest. This is possible since the process was stopped as soon as k reached a value greater than $(1-\beta)/(\beta-\alpha)$ which means that $k - 1 \leq (1-\beta)/(\beta-\alpha)$. Thus, the optimal algorithm can travel the necessary distance of βi by the time the i^{th} set expires at time $\alpha i + 1$ for any $i \leq k \leq 1 + (1-\beta)/(\beta-\alpha)$. (Note that the required condition is that $i \leq 1/(\beta-\alpha)$. The fact that $i \leq 1 + (1-\beta)/(\beta-\alpha)$ is equivalent to $i \leq (1-\alpha)/(\beta-\alpha)$. Since $\alpha > 0$, the condition holds.) Thus, the optimal can serve $c + c^2 + \cdots + c^k$ requests. The algorithm gets $1 + c + c^2 + \cdots + c^{k-1}$. This results in a competitive ratio of c.

Since c was chosen to be an arbitrary constant, the competitive ratio is unbounded. □

The remaining two lower bounds hold even when $\alpha = 1$ which means that the online algorithm does not need to notify a request as to whether it will be serviced. The following lower bound shows that the bound proven for Double-Gain is within a constant factor of the optimal competitive ratio.

Theorem 8 *For any α and any $\beta \geq 1/2$, there is a metric space with diameter β on which no deterministic algorithm can achieve a better competitive ratio than*

$$\frac{\beta}{1-\beta}\left\lceil \frac{1}{1-\beta} \right\rceil .$$

Proof. The metric space will have $n = mk$ points which will consist of m groups, each with k points. Any two points from the same group are a distance of $1-\beta$ apart. Any two points from different groups are β apart.

In every interval of length $1-\beta$, there will be an *arrival event* in which a request will arrive on each point in the metric space, with the following exceptions. If the online server is located at a point in a particular group or moving between two points within the same group at the time of an arrival event, then no request will arrive at any point within the group at that time. If the online server is moving between two points from different groups at the time of an arrival event, then no request will arrive on any of the points in the group that the server is moving away from or the group that the server is moving towards. In these cases, we say that a group has been *skipped*.

Once the online server reaches a group, it will only be able to serve one request. This is because no requests will have arrived on points in that group during the interval of β in which the server is moving towards the group. There is exactly one arrival event in the $1-\beta$ time preceding the time the server starts moving towards the group. All previous requests will have expired by the time the server reaches the group. The server will serve one request upon its arrival. By the time it travels the $1-\beta$ distance to reach another point in the group, all outstanding requests in the group will have expired. Thus, the online server serves at most one request every interval of length β.

The optimal server will stay at a group until that group is skipped. That is, it will stay through the last arrival event before the group is skipped and then go to another group. The server will plan its departure so that it arrives at the new group just after an arrival event. Thus, the server will be in transit between groups for $\lceil \beta/(1-\beta) \rceil$ arrival events.

Let's say that the server leaves a group at time t. It will select a new group to visit which was not skipped in the arrival event preceding time t and will be the last group to be skipped after time t. Since at most two groups are skipped every arrival event, the server will find a group at which it will stay for $(m-2)/2$ arrival events after time t. Thus, the optimal server will be serving requests for $(m-2)/2 - \lceil \beta/(1-\beta) \rceil$ arrival events every $(m-2)/2$ arrival events.

The optimal server moves from point to point within a group every $1-\beta$ time units, arriving at each point just after an arrival event. When it arrives at a point, it can serve all the $\lceil 1/(1-\beta) \rceil$ requests which are waiting at that point and have not yet expired. Then it moves to another point in the group and serves all the requests waiting at that point. As long as k, the number of points in a group is at least $\lceil 1/(1-\beta) \rceil$, it can serve $\lceil 1/(1-\beta) \rceil$ requests every interval of length $1-\beta$ unless it happens to be in transit between groups. Thus, the rate at which it serves requests is

$$\frac{1}{1-\beta}\left\lceil \frac{1}{1-\beta} \right\rceil \left(1 - \left\lceil \frac{\beta}{1-\beta} \right\rceil \frac{2}{m-2}\right).$$

As m gets large, this number gets arbitrarily close to

$$\frac{1}{1-\beta}\left\lceil \frac{1}{1-\beta} \right\rceil.$$

Putting this together with the upper bound on the rate of service for the online algorithm, no algorithm can have a competitive ratio better than

$$\frac{\beta}{1-\beta}\left\lceil \frac{1}{1-\beta} \right\rceil.$$

□

The following lower bound gives a stronger bound for the case where $\beta < 1/2$.

Theorem 9 *For any α and any $\beta < 1$, there is a metric space with diameter β on which no deterministic online algorithm can obtain a competitive ratio less than* 2.

Proof. Let $N = 1 + \lceil 1/\beta \rceil$. Consider a uniform metric space with $6N$ points, each spaced β apart. The adversary will have an arrival event every interval of length β. The i^{th} arrival event takes place at time βi. At the i^{th} arrival event, a request will arrive at node $i \bmod 3N$ and $3N + (i \bmod 3N)$. The time interval between two requests arriving at the same node is $3N\beta$ which is greater than 3.

Consider the i^{th} arrival event. Both of the requests for this arrival event expire at time $1 + \beta i$. The adversary will plan on satisfying exactly one of these requests for this arrival event at time $1 + \beta i - \epsilon$, where $\epsilon << \beta$. The adversary will select one of the two nodes $i \bmod 3N$ or $3N + (i \bmod 3N)$ and satisfy the request at this node. The adversary will select the node that the algorithm is farthest away from at time $1 + \beta i - \epsilon$. This means that the algorithm is at least a distance of $\beta/2$ from the selected node at time $1 + \beta i - \epsilon$.

The adversary will also arrange to have a request arrive at the selected node at time $1 + \beta i - \epsilon$. We will call these requests *additional* requests. This ensures that the adversary can satisfy two requests every interval of length β.

We will now show that the online algorithm can never serve two requests within an interval of length less than β. Suppose for purposes of contradiction that the online algorithm does serve two requests within an interval of length less than β. These two requests can not be at different nodes since it takes time β to travel from one node to another, so the two requests served must be at the same node x. This implies that there are two active requests at node x within an interval of length less than β in which the online algorithm also has its server at node x.

Aside from the additional requests, the period between request arrivals at any node is greater than 3 which means that one request expires at least two time units before the next one arrives. Thus, the only hope of having two active requests at the same node within an interval of length less than β is if one of the requests is an additional request. Consider such an example. Request 1 arrives at time βi. Request 2 arrives at time $\beta i + 1 - \epsilon$ and the next request (Request 3) arrives after time $\beta i + 3$. Request 2 expires at least one time unit before Request 3 arrives, so the requests served by the online algorithm at node x must be Requests 1 and 2. By the condition of having an additional request, we know that the algorithm is a distance of at least $\beta/2$ away from node x at time $1 + \beta i - \epsilon$. Thus, we are guaranteed that the algorithm is never on node x during the interval $[1 + \beta(i - 1/2) - \epsilon, 1 + \beta(i + 1/2) - \epsilon)$. Thus, Request 1 must be served before time $1 + \beta(i - 1/2) - \epsilon$ and Request 2 must be served after time $1 + \beta(i + 1/2)) - \epsilon$. This contradicts the assertion that the two requests are served within an interval of length less than β. □

7 Conclusions and Open Questions

Naturally, it is the goal of this research to find the optimal competitive ratio which can be obtained on any metric space as a function of α and β. We believe that a stronger upper bound can be obtained for the algorithm Double-Gain. It would also be interesting to extend the bound for Double-Gain to the case of non-uniform job windows. Another natural algorithm to consider is the BATCH algorithm with insertions, where the algorithm can service a newly arrived request or a request which is about to expire if it can be handled without having to drop any of the requests in the current BATCH path.

Another important extension of this work would be to incorporate non-zero service times.

References

[1] H. Alborzi, E. Torng, P. Uthaisombut and S. Wagner. The k-client Problem. *Journal of Algorithms*, vol.41, no. 2, Nov. 2001, pp.115-73.

[2] N. Ascheuer, S.O. Krumke and J. Rambau. Online dial-a-ride problems: minimizing the completion time. In *the Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, vol. 1770, Springer-Verlag, 2000, pp.639-50.

[3] J. Augustine and S. Seiden. Linear Time Approximation Schemes for Vehicle Routing. To appear in the 8^{th} *Scandinavian Workshop on Algorithm Theory*, 2002.

[4] B. Awerbuch, Y. Azar, A. Blum and S. Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, vol. 28, no. 1, 1998. pp.254-62.

[5] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie and M. Talamo. Algorithms for the On-line Traveling Salesman. *Algorithmica*, vol. 29, no. 4, April 2001, pp. 560-81.

[6] D. Bertsimas and G. Van Ryzin. Stochastic and Dynamic Vehicle Routing in the Euclidean Plane with Multiple Capacitated Vehicles. *Operations Research*, vol. 41, no. 1, Jan.-Feb. 1993, pp.60-76.

[7] D. Bertsimas and G. Van Ryzin. A Stochastic and Dynamic Vehicle Routing Problem in the Euclidean Plane. *Operations Research*, vol. 39, no. 4, July-Aug. 1991, pp.601-615.

[8] X. Deng and C. Papadimitriou Exploring an unknown graph. In *the Proceedings of 31st Annual Symposium on Foundations of Computer Science*, 1990, pp.355-61.

[9] E. Feuerstein and L. Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, vol. 268, no. 1, Oct. 2001, pp.91-105.

[10] D. Hauptmeier, S.O. Krumke and J. Rambau. The online dial-a-ride problem under reasonable load. In *the Proceedings of Algorithms and Complexity, 4th Italian Conference*, Lecture Notes in Computer Science, vol. 1767, Springer-Verlag, 2000, pp.125-36.

[11] D. Hauptmeier, S.O. Krumke, J. Rambau and H.C. Wirth. Euler is standing in line dial-a-ride problems with precedence-constraints. *Discrete Applied Mathematics*, vol. 113, no. 1, Sept. 2001, pp.87-107.

[12] P. Jaillet. A priori Solution of a Traveling Salesman Problem in Which a Random Subset of the Customers Are Visited. *Operations Research*, vol. 36, no. 6, Nov.-Dec. 1988, pp.929-936.

[13] S.O. Krumke, W.E. de Paepe, D. Poensgen and L. Stougie. News from the Online Traveling Repairman. In *the Proceedings of the 26th International Symposium on Mathematical Foundations*

of Computer Science, Lecture Notes in Computer Science, vol. 2136, Springer-Verlag, 2001, pp.487-499.

[14] M. Manasse, L.A. McGeogh and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, vol. 11, no. 2, 1990, pp.208-230.

[15] W. Powell, P. Jaillet and A. Odoni. Stochastic and dynamic networks and routing. *Network Routing*, M. Ball, T. Maganti, C. Monma and G. Nemhauser (eds.). North-Holland, Amsterdam, 1995.

[16] H. Psarafits. Dynamic Vehicle Routing Problems. *Vehicle Routing: Methods and Studies*, B.L. Golden and A.A. Assad (eds.). North-Holland, Amsterdam, 1988.