Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

SVEN O. KRUMKE

News from the Online Traveling Repairman

ZIB-Report 00-08 (March 2000)

NEWS FROM THE ONLINE TRAVELING REPAIRMAN

SVEN O. KRUMKE [1]

ABSTRACT. The traveling repairman problem (TRP) is a variant of the famous traveling salesman problem (TSP). The objective for the TRP is to minimize the latency, that is, the the weighted sum of completion times of the cities, where the completion time of a city is defined to be the time in the tour before the city is reached.

In the online traveling repairman problem OLTRP requests for visits to cities (points in a metric space) arrive online while the repairman is traveling. We analyze the performance of algorithms using competitive analysis, where the cost of an online algorithm is compared to that of an optimal offline algorithm. An optimal offline algorithm knows the entire request sequence in advance and can serve it with minimum cost.

Recently, Feuerstein and Stougie [FS00] presented a 9-competitive algorithm for the OLTRP on the real line. In this paper we show how to use techniques from online-scheduling to obtain an 8-competitive deterministic algorithm which works for any metric space. We also present a randomized algorithm which has a competitive ratio of $\frac{4}{\ln 2} \approx 5.7708$ against an oblivious adversary. All of our results also hold for the "dial-a-ride" generalization of the OLTRP, where objects have to be picked up and delivered by a server.

1. INTRODUCTION

The *traveling repairman problem* (TRP) is a variant of the famous traveling salesman problem. Given a set of n points $p_1, \ldots, p_n$ in a metric space and a tour which visits the points in some order, define the completion time C_j of a point p_j (in this tour) to be the length of the tour from the origin before it reaches p_j. Given weights $w_1, \ldots, w_n$ for the points, the goal of the TRP is to minimize the average weighted completion time or equivalently the total weighted completion time, i.e., the objective function $\sum_{j=1}^{n} w_j C_j$. This objective function is also referred to as the *latency*.

In this paper we consider the following online variant of the TRP called the *online traveling repairman problem* (OLTRP). Requests to visits for points arrive over time while the repairman is traveling. The repairman moves at unit speed through the metric space starting its work at a designated origin. A request is considered served when its corresponding point in the metric space is visited by the repairman but not earlier than its release time. The completion time of a request is the time in the tour before the request is served. In this model the server is allowed to wait at the cost of time that elapses. Decisions are revocable as long as they have not been executed. Only history is irrevocable. The objective for the OLTRP is to minimize the weighted sum of completion times.

[1]Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization, Takustr. 7, D-14195 Berlin-Dahlem, Germany. Email: krumke@zib.de. Research supported by the German Science Foundation (DFG, grant Gr 883/5-3)

Key words and phrases. Traveling Repairman, Latency, Dial-a-Ride-Problem, Competitive Analysis.

An online algorithm does not know about the existence of a request before its release time. A common way to evaluate the quality of online algorithms is *competitive analysis* [BEY98]: An algorithm is called c-competitive if its cost on any input sequence is at most c-times the cost of an optimal offline algorithm.

The two main results of this paper are an 8-competitive deterministic algorithm and a randomized algorithm which achieves a competitive ratio of $\frac{4}{\ln 2} \approx 5.7708$ against an oblivious adversary. These algorithms are adoptions of the GREEDY-INTERVAL algorithm for online-scheduling presented in [HSW96, HS$^+$97] and of the randomized version given in [CP$^+$96].

For the case of the real line our results improve previous best known upper bounds for the competitive ratio of algorithms for the OLTRP. Moreover, for the case of general metric spaces our algorithms are the first competitive algorithms. All of our results continue to hold for the "dial-a-ride" generalization (called OLLDARP) of the OLTRP. In fact, we present the results for the generalized version. In the dial-a-ride problem each request consists of an object which has to be picked up and delivered at its source and destination, respectively. Preemption is not allowed: once the server has picked up an object, it is not allowed to drop it at any other place than its destination.

1.1. **Previous Work.** Feuerstein and Stougie [FS00] present a 9-competitive algorithm for the OLTRP on the real line and a 15-competitive algorithm for OLLDARP on the real line for the case that the server has infinite capacity. Feuerstein and Stougie also prove lower bounds of $1 + \sqrt{2}$ and 3 for the competitive ratio of any deterministic algorithm for OLTRP and OLLDARP, respectively, on the real line.

The problem of minimizing the makespan was considered in [AF$^+$99] for the case of the Online-TSP. Online dial-a-ride problems with this objective were studied in [AKR00, FS00].

1.2. **Outline.** In Section 2 we formally define the problems under study. Section 3 contains the deterministic algorithm INTERVAL and the proof of its competitiveness. In Section 4 we present the randomized algorithm RANDINTERVAL which achieves an improved competitive ratio. In that section we also show how to apply the randomization techniques from algorithm RANDINTERVAL to obtain an extremely simple algorithm RANDSLEEP for the problem of minimizing the makespan. Although the competitive ratio of RANDSLEEP, which equals $1 + 1/\ln 2 \approx 2.4427$, does not beat the best known bound of 2 for the makespan problem, the proof of the performance is very simple. The competitive ratios for the algorithms and the corresponding lower bounds are given in Table 1.

TABLE 1. Competitive ratios for the problems OLTRP and OLLDARP.

Algorithm	INTERVAL	RANDINTERVAL	Previous best	Lower bound
OLTRP	8	$4/\ln 2$	9 [FS00]	$1 + \sqrt{2}$ [FS00]
OLLDARP	8	$4/\ln 2$	—	3 [FS00]

2. PRELIMINARIES

An instance of the online dial-a-ride problem OLLDARP consists of a metric space $M = (X, d)$ with a distinguished origin $o \in X$ and a sequence $\sigma = r_1, \ldots, r_m$ of requests. It is assumed that for all pairs of points (x, y) from M, there is a path $p : [0, 1] \to X$ in X with $p(0) = x$ and $p(1) = y$ of length $d(x, y)$ (see [AF$^+$99] for a thorough discussion of this model). Examples of a metric spaces that satisfy the above condition are the Euclidean space $\mathbb{R}^p$ and a metric space induced by an undirected edge-weighted graph.

Each request is a quadruple $r_i = (t_i, w_i, a_i, b_i)$, where $t_i \geq 0$ is a real number, the time where request r_i is released (becomes known), $w_i \geq 0$ is its weight factor (used in the objective function), and $a_i, b_i \in V$ are the source and destination, respectively, between which the new object is to be transported. We assume that the sequence $\sigma = r_1, \ldots, r_m$ of requests is given in order of non-decreasing release times.

A server is located at the origin $o \in X$ at time 0 and can move at constant unit speed. The server has capacity $C \in \mathbb{N} \cup \{\infty\}$, i.e., it can carry at most C objects at a time. We do not allow *preemption*: once the server has picked up an object, it is not allowed to be dropped at any other place than its destination. The online traveling repairman problem OLTRP is obtained as the special case of the OLLDARP when for each request its source and destination coincide (regardless of the capacity of the server).

An online algorithm for OLLDARP does neither have information about the release time of the last request nor about the total number of requests. The online algorithm must determine the behavior of the server at a certain moment t in time as a function of all the requests released up to time t (an of the current time t). In contrast, an offline algorithm has information about all requests in the whole sequence σ already at time 0.

Given a sequence σ of requests, a *valid transportation schedule* for σ is a sequence of moves of the server such that the following conditions are satisfied: (a) The server starts its movement in the origin vertex o, (b) each transportation request in σ is served, but starting not earlier than its release time.

The objective function of the OLLDARP is the *latency*, which is defined to be the *weighted sum of completion times*: Let C_j denote the time when request r_j completes in the schedule produced by some algorithm ALG. Then the latency of the solution of ALG on σ is $\mathsf{ALG}(\sigma) := \sum_{j=1}^{n} w_j C_j$. We use OPT to denote an optimal offline algorithm.

An online algorithm ALG for OLLDARP is *c-competitive*, if there exists a constant c such that for any request sequence σ:

$$\mathsf{ALG}(\sigma) \leq c \cdot \mathsf{OPT}(\sigma).$$

If ALG is a randomized algorithm, then $\mathsf{ALG}(\sigma)$ must be replaced by the expected cost $\mathrm{E}[\mathsf{ALG}(\sigma)]$, where the expectation is taken over the random choices made by ALG. In this paper we evaluate randomized online algorithms only against an *oblivious adversary*. An oblivious adversary must generate a request sequence in advance and is not allowed to see the random choices made by the online algorithm. We refer to [BEY98] for details on the various adversary models. A randomized online algorithm ALG is c-competitive against an oblivious adversary if for any request sequence σ:

$$\mathrm{E}[\mathsf{ALG}(\sigma)] \leq c \cdot \mathsf{OPT}(\sigma).$$

3. A Deterministic Algorithm

Our deterministic strategy is an adaption of the algorithm GREEDY-INTERVAL presented in [HSW96, HS+97] for online-scheduling. The modifications in the algorithm are minor and the proof of the performance uses the same techniques.

Strategy INTERVAL: We first describe the initialization phase of the algorithm, that is, the initial behavior of the algorithm, before schedules are actually computed or followed.

If there is no request released at time 0, the server waits until the time t_1, where the first request is released. We then set $L := t_1$.

If requests R are released at time 0, compute the minimum time T by that the first of the requests in R could be completed. In what follows we assume that $T > 0$, since in the case $T = 0$ there are requests with source and destination 0 released at time 0 and these requests can be served at no cost by the algorithm.

If no further requests are released before time T, the server waits until time T and we set $L := T$. Otherwise, if requests are released at some time t with $0 < t < T$, then we set $L := t$. This completes the initialization.

After the initialization the algorithm works in phases as follows. For $i = 1, 2, \ldots$ let $B_i := 2^{i-1}L$. We also set $B_0 := L/2$. For $i \geq 1$, the ith phase starts at time B_i. At time B_i the algorithm considers all the requests released up to time B_i but not yet scheduled (in any of the previously computed schedules). Call the set of those requests R_i. The algorithm now computes a schedule for the requests in R_i with the following properties:

(i) The schedule starts and ends in the origin o.
(ii) The schedule has length at most $2B_i = B_{i+1}$.
(iii) The schedule maximizes the weight of requests served among all schedules satisfying (i) and (ii).

The schedule computed this way is set to be followed from time B_{i+1} until B_{i+2} (Since $B_{i+2} - B_{i+1} = (2^{i+1} - 2^i)L = 2^iL = B_{i+1}$, the schedule can actually be completed in the designated time period).

We now analyze the competitiveness of algorithm INTERVAL. Notice that the number L computed in the initialization phase is clearly a lower bound on the completion time of any request in an optimal offline solution. We will need this later.

Lemma 3.1. *Let S_i be the set of requests scheduled in phase $i \geq 1$ of Algorithm* INTERVAL *and denote by S_i^* denote the set of requests in the optimal offline schedule that complete in the time interval $]B_{i-1}, B_i]$. Then*

$$\sum_{i=0}^{k} w(S_i) \geq \sum_{i=0}^{k} w(S_i^*) \quad \text{for } k = 1, 2, \ldots . \tag{1}$$

Proof. Consider the kth phase of the algorithm. A schedule is computed at time $B_k = 2^{k-1}L$. Let $S := \bigcup_{i=1}^{k} S_k^* \setminus \bigcup_{i=1}^{k-1} S_k$ be the requests in the schedule of OPT that complete by time B_k but have not yet been scheduled by INTERVAL. Clearly, none of those requests can have been released before time B_k. Thus, all of them are available for planning at time B_k for INTERVAL.

Consider the optimum offline schedule which starts in the origin o but not necessarily ends in o. If we follow the first B_k time units of this schedule (with the modification that if an object is picked up but not delivered before time B_k we omit this pickup action) and then return to the origin, we obtain a schedule of length at most $2B_k$ delivering all requests in $\bigcup_{i=1}^{k} S_k^*$ and in particular all requests in S. Since this schedule is one possibility for INTERVAL in phase k, it follows that the weight of requests scheduled in iterations 1 to k is at least that of $\bigcup_{i=1}^{k} S_k^*$, which means that inequality (1) holds. □

The previous lemma gives us the following bound on the number of phases that INTERVAL uses to process a given input sequence σ.

Corollary 3.2. *Suppose that the optimum offline schedule is completed at time $T \in]B_{p-1}, B_p]$ for some $p \geq 1$. Then, the number of phases of Algorithm* INTERVAL *is at most p. The schedule of* INTERVAL *is completed no later than time B_{p+2}.*

Proof. By Lemma 3.1 the weight of all requests scheduled in the first p phases equals the total weight of all requests. Hence all requests must be scheduled within the first p phases. Since, by construction of INTERVAL, the schedule computed in phase p completes by time B_{p+2}, the claim follows. □

To prove the performance result for INTERVAL we need one more elementary lemma.

Lemma 3.3. *Let $a_i, b_i \in \mathbb{R}_{\geq 0}$ for $i = 1, \ldots, p$ and suppose that the following two conditions are satisfied:*

(i) $\sum_{i=1}^{p} a_i = \sum_{i=1}^{p} b_i$*;*
(ii) $\sum_{i=1}^{p'} a_i \geq \sum_{i=1}^{p'} b_i$ *for all* $1 \leq p' \leq p$*.*

Then the inequality

$$\sum_{i=1}^{p} \tau_i a_i \leq \sum_{i=1}^{p} \tau_i b_i \tag{2}$$

holds for any nondecreasing sequence $0 \leq \tau_1 \leq \cdots \leq \tau_p$.

Proof. Straightforward induction by p. □

Theorem 3.4. *Let $\sigma = r_1, \ldots, r_m$ be any sequence of requests. Denote by $C_j^* \in]B_{\phi_j}, B_{\phi_{j+1}}]$ the completion time of request r_j in the optimal schedule* OPT(σ)*. Then*

$$\text{INTERVAL}(\sigma) \leq 8 \sum_{j=1}^{n} w_j B_{\phi_j}. \tag{3}$$

Proof. By definition of INTERVAL, each request scheduled in phase i completes no later than time B_{i+2}. Summing up over all phases $1, \ldots, p$ we get that

$$\text{INTERVAL}(\sigma) \leq \sum_{i=1}^{p} B_{i+2} w(S_i) = 8 \sum_{i=1}^{p} B_{i-1} w(S_i). \tag{4}$$

Applying Lemma 3.3 to the sequences $w(S_i)$ and $w(S_i^*)$ with the weighing sequence $\tau_i := B_{i-1}$ we get that the right and side of (4) is bounded by

$$8\sum_{i=1}^{p} B_{i-1}w(S_i^*) = 8\sum_{j=1}^{n} B_{\phi_j}w_j.$$

This proves the claim. □

Since the completion time C_j^* of request σ_j in the optimum schedule satisfies $C_j^* > B_{\phi_j}$, it follows immediately from Theorem 3.4 that the total weighted sum of completion times produced by INTERVAL is at most 8 times the optimum value. We thus obtain the following theorem.

Theorem 3.5. *Algorithm* INTERVAL *is* 8*-competitive for* OLLDARP. □

4. AN IMPROVED RANDOMIZED ALGORITHM

In this section we show how to use techniques from [CP+96] to obtain an algorithm RANDINTERVAL with improved competitive ratio.

At the beginning, the algorithm RANDINTERVAL chooses a random number $x \in]0,1]$ according to the uniform distribution. From this moment on the algorithm is completely deterministic, working like the deterministic algorithm INTERVAL presented in the last section. The only difference is that the first phase starts at time L (we set $B'_1 := 2^{-x}L$) and for $i \geq 2$ the ith phase starts at time $B'_i := 2^{i-1-x}L$ instead of at time $B_i = 2^{i-1}$. For $i \geq 1$ the schedule computed in the ith phase is followed from time B'_{i+1} to B'_{i+2}. Notice that for any random choice of $x \in]0,1]$ the starting time $2^{1-x}L$ of the schedule computed in the first phase is not earlier than the time L where this phase starts. Thus, in fact, the algorithm produces a feasible schedule.

By the proof of Theorem 3.4 it follows that for a sequence $\sigma = r_1, \ldots, r_m$ of requests the expected objective function value of RANDINTERVAL satisfies:

$$\mathrm{E}\left[\textsc{RandInterval}(\sigma)\right] \leq \mathrm{E}\left[8\sum_{i=1}^{n} B'_{\phi_j}w_j\right] = 8\sum_{i=1}^{n} w_j \mathrm{E}\left[B'_{\phi_j}\right], \tag{5}$$

where $]B'_{\phi_j}, B'_{\phi_{j+1}}]$ is the interval containing the completion time C_j^* of request r_j in the optimal solution $\textsc{Opt}(\sigma)$. To prove a bound on the performance of RANDINTERVAL we now compute $\mathrm{E}\left[B'_{\phi_j}\right]$. Notice that B_{ϕ_j} is the largest value $2^{k-x}L$ in the set $\{2^{-x}L, 2^{1-x}L, 2^{2-x}L, \ldots\}$ which is strictly smaller than C_j^*.

Lemma 4.1. *Let $z \geq L$ be a real number and let $x \in]0,1]$ be randomly chosen according to the uniform distribution on $]0,1]$. Define the random variable B by*

$$B := \max\{\, 2^{k-x}L : 2^{k-x}L < z \text{ and } k \in \mathbb{N}\,\}$$

Then, the expected value $E[B]$ of B satisfies

$$E[B] = \frac{z}{2\ln 2}.$$

Proof. Suppose that $2^k L \leq z < 2^{k+1} L$ for some $k \geq 0$. Observe that

$$B = \begin{cases} 2^{k-x} L & \text{if } x \leq \log_2 \frac{2^{k+1} L}{z} \\ 2^{k+1-x} L & \text{otherwise} \end{cases}$$

Hence

$$\begin{aligned} \mathrm{E}[B] &= \int_0^{\log_2 \frac{2^{k+1} L}{z}} 2^{k-x} L \, dx + \int_{\log_2 \frac{2^{k+1} L}{z}}^1 2^{k+1-x} L \, dx \\ &= \int_0^{\log_2 \frac{2^{k+1} L}{z}} 2^{k-x} L \, dx + \int_{\log_2 \frac{2^{k+1} L}{z}}^1 2^{k-x} L \, dx + \int_{\log_2 \frac{2^{k+1} L}{z}}^1 2^{k-x} L \, dx \\ &= L 2^k \int_0^1 2^{-x} \, dx + L 2^k \int_{\log_2 \frac{2^{k+1} L}{z}}^1 2^{-x} \, dx \\ &= L 2^k \left[-\frac{1}{\ln 2} 2^{-x} \right]_0^1 + L 2^k \left[-\frac{1}{\ln 2} 2^{-x} \right]_{\log_2 \frac{2^{k+1} L}{z}}^1 \\ &= \frac{z}{2 \ln 2}. \end{aligned}$$

This completes the proof. □

From Lemma 4.1 we can conclude that $\mathrm{E}\left[B'_{\phi_j}\right] = \frac{1}{2 \ln 2} C^*_j$. Using this result in inequality (5) yields the following theorem:

Theorem 4.2. *Algorithm* RANDINTERVAL *is c-competitive with $c = \frac{4}{\ln 2} \approx 5.7708$ for* OLLDARP *with respect to the objective function latency against an oblivious adversary.* □

4.1. **A Randomized Algorithm for the Minimization of the Makespan.** We close this section by showing how the randomization technique presented above can be used to obtain an algorithm for the Online Dial-a-Ride Problem when the objective function is the *makespan*. Call this version OLDARP. We use the setting of [FS00, AKR00], where the server has to return to the origin after serving all requests. The makespan is then defined to be the the time when the server is back at the origin after all requests have been served.

The algorithm RANDSLEEP presented below has a competitive ratio of $1+1/\ln 2 \approx 2.4427$ against an oblivious adversary. Although the best deterministic algorithm called SMARTSTART is 2-competitive [AKR00], RANDSLEEP beats the competitive ratio of $5/2$ achieved by the algorithms IGNORE and REPLAN(see [FS00, AKR00] for the definitions of these algorithms and proofs of their performance). Moreover, the beauty of RANDSLEEP are its simplicity and the easy proof of its performance.

Strategy RANDSLEEP: At the beginning, the algorithm chooses a random number $x \in]0, 1]$ according to the uniform distribution. After this random choice, the algorithm is again completely deterministic.

We first compute a lower bound L on the optimal offline makespan. If there are requests released at time 0 then we let L be the makespan of the shortest schedule serving all these requests which starts and ends in the origin o (as in Algorithm INTERVAL we can assume that in this case $L > 0$). If no requests

are released at time 0, then L is set to the release time of the first request. This completes the initialization.

Again the algorithm works in phases. For $i = 1, 2, \ldots$ let $B_i := 2^{i-x}L$. Phase i is started at time B_i. At this time the algorithm considers all requests R_i that have been released up to time B_i but not been served yet. RANDSLEEP computes a shortest schedule for all requests in R_i which starts and ends in the origin. If this schedule can be completed no later than time B_{i+1}, the server follows this schedule. If on the other hand the schedule needs more than $B_{i+1} - B_i = B_i$ time units, then the server simply does nothing: it sleeps until time B_{i+1}.

Theorem 4.3. *Algorithm* RANDSLEEP *is c-competitive for* OLDARP *with respect to the makespan, where $c = 1 + 1/\ln 2 \approx 2.4427$.*

Proof. Let $Z^* = \mathsf{OPT}(\sigma)$ denote the optimum makespan for the input sequence σ and suppose that $2^{k-x}L < Z^* \leq 2^{k+1-x}L$ for some $k \geq 0$. Notice that such a $k \geq 0$ must exist, since L is a lower bound on Z^* and $x > 0$. By Lemma 4.1 we have that $\mathrm{E}\left[2^{k-x}L\right] = Z^*/(2\ln 2)$. Since the optimum schedule can be completed before time $2^{k+1-x}L$, we can conclude two facts: first, all requests have been released by time $2^{k+1-x}L$, and second, in the $(k+1)$st phase RANDSLEEP must have scheduled all requests, since in this phase it allows $2^{k+1-x}L$ time units for a schedule serving all unserved requests. The length of the schedule computed in the $(k+1)$st phase can not exceed Z^*. Hence

$$\mathrm{E}\left[\mathsf{RANDSLEEP}(\sigma)\right] \leq 4\mathrm{E}\left[2^{k-1-x}L\right] + Z^* = \left(1 + \frac{1}{\ln 2}\right) Z^*.$$

This is what we wanted to show. □

REFERENCES

[AF+99] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo, *Algorithms for the on-line traveling salesman*, Algorithmica (1999), To appear.

[AKR00] N. Ascheuer, S. O. Krumke, and J. Rambau, *Online dial-a-ride problems: Minimizing the completion time*, Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 1770, Springer, 2000, pp. 639–650.

[BEY98] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.

[CP+96] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J Wein, *Improved scheduling algorithms for minsum criteria*, Proceedings of the 23rd International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 1099, Springer, 1996, pp. 646–657.

[FS00] E. Feuerstein and L. Stougie, *On-line single server dial-a-ride problems*, Theoretical Computer Science (2000), To appear.

[HS+97] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line approximation algorithms*, Mathematics of Operations Research **22** (1997), 513–544.

[HSW96] L. Hall, D. B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line algorithms*, Proceedings of the 7thAnnual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 142–151.

SVEN O. KRUMKE, KONRAD-ZUSE-ZENTRUM FÜR INFORMATIONSTECHNIK BERLIN, DEPARTMENT OPTIMIZATION, TAKUSTR. 7, D-14195 BERLIN-DAHLEM, GERMANY.
E-mail address: krumke@zib.de
URL: http://www.zib.de/krumke