Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

NORBERT ASCHEUER SVEN O. KRUMKE JÖRG RAMBAU

The Online Transportation Problem: Competitive Scheduling of Elevators

Preprint SC 98-34 (December 1998)

THE ONLINE TRANSPORTATION PROBLEM: COMPETITIVE SCHEDULING OF ELEVATORS

NORBERT ASCHEUER, SVEN O. KRUMKE, AND JÖRG RAMBAU

ABSTRACT. In this paper we consider the following online transportation problem (OLTP): Objects are to be transported between the vertices of a given graph. Transportation requests arrive online, specifying the objects to be transported and the corresponding source and target vertex. These requests are to be handled by a server which commences its work at a designated origin vertex and which picks up and drops objects at their starts and destinations. After the end of its service the server returns to its start. The goal of OLTP is to come up with a transportation schedule for the server which finishes as early as possible.

We first show a lower bound of $5/3$ for the competitive ratio of any deterministic algorithm. We then analyze two simple and natural strategies which we call Replan and Ignore. Replan completely discards its schedule and recomputes a new one when a new request arrives. Ignore always runs a (locally optimal) schedule for a set of known requests and ignores all new requests until this schedule is completed.

We show that both strategies, Replan and Ignore, are $5/2$-competitive. We also present a somewhat less natural strategy Sleep which in contrast to the other two strategies may leave the server idle from time to time although unserved requests are known. We also establish a competitive ratio of $5/2$ for the algorithm Sleep.

Our results are extended to the case of "open schedules" where the server is not required to return to its start position at the end of its service.

1. INTRODUCTION

Transportation problems where objects are to be transported between given sources and destinations in a metric space are classical problems in combinatorial optimization. In the classical setting, one assumes that the complete input for an instance is available for an algorithm to compute a solution. In many cases this *offline optimization* does not reflect the real-world situation appropriately. For instance, the transportation requests in an elevator system are hardly known in advance. Decisions have to be made *online* without the knowledge of future requests.

Online algorithms are tailored to cope with such situations. They work on request sequences and, as soon as a new request arises, update their solution to serve the new request as well. A common way to evaluate the quality of online algorithms is *competitive analysis* [BEY98, FW98].

In this paper we consider the following online transportation problem (OLTP): Objects are to be transported between the vertices of a given graph. A request consists of the objects to be transported and the corresponding source and target vertex of the transportation request. The requests arrive online and must be handled by a server which commences and ends its work at a designated origin vertex and which moves

Key words and phrases. Vehicle Routing, Online-Algorithms, Competitive Analysis.

Research supported by the German Science Foundation (DFG, grant Gr 883/5-1).

along the paths in the graph. The server picks up and drops objects at their starts and destinations. We assume that neither the release time of the last request nor the number of requests is not known in advance. The goal of OLTP is to come up with a transportation schedule for the server which finishes as early as possible.

Our investigations of the OLTP were originally motivated by the performance analysis of a large distribution center of Herlitz AG, Berlin [AG+98]. Its automatic pallet transportation system employs several vertical transportation systems (elevators) in order to move pallets between the various floors of the building. The pallets that have to be transported during one day of production are not known in advance. If the objective is chosen as minimizing the completion time (makespan) then this can be modeled by the OLTP where the underlying graph is a path. All of our algorithms solve "offline instances" of OLTP during their run. On general graphs this task is NP-hard, since it contains the Hamiltonian path problem as a special case [Fre93]. On paths, however, one can solve the static transportation problem efficiently (by an easy modification of the algorithm presented in [AK88]).

2. PRELIMINARIES

An instance of the online transportation problem OLTP consists of an undirected graph $G=(V,E)$ with edge-weights $d(e)$ $(e \in E)$ and a distinguished origin vertex $o \in V$.

Each request is a triple $r_i = (t_i, a_i, b_i)$, where t_i is a real number, the time where request r_i becomes known, and $a_i \in V$ and $b_i \in V$ are the source and target, respectively, between which the new object is to be transported. We will consider each set $\sigma = \{r_1, \dots, r_m\}$ of transportation requests as ordered by the release times. We assume that the online algorithm does neither have information about when the last request arrives nor about the total number of requests.

The server can move at constant unit speed and is located at the start o at time 0. We allow the server to move "continuously" on the graph, i.e., to move continuously from one end point of an edge (u,v) to the other one and possibly change its direction while at some location s on the edge (u,v). The server has *unit-capacity*, i.e., it can carry at most one object at a time. Finally, we do *not allow preemption*: once the server has picked up an object, it is not allowed to drop it at any other place than its destination.

It is easy to see that in the setting above, without loss of generality one can restrict the server to move along shortest paths in the graph. Thus, we will not state explicitly which path the server actually takes and just say that it moves from a point x to a point y. Hence, we can assume that the underlying graph G is complete with edge-weights satisfying the triangle inequality. In the sequel we use $d(a,b)$ to denote the shortest path distance between the vertices a and b in the graph G. We also extend the distance function d to the infinitely many points that lie on the edges of G: If s is on edge (u,v) at distance $d(s,u)$ from u and distance $d(s,v) = d(u,v) - d(s,u)$ from s, then its distance from any other point x on the graph is given by $\min\{d(s,u) + d(u,x), d(s,v) + d(v,x)\}$.

Given a set 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 the time it becomes known, and (c) the server returns to the origin vertex after having served the last request.

Let $C_{\mathsf{A}}(\sigma)$ denote the completion time of the server moved by algorithm A on the set σ of requests. We also use OPT to denote the optimal offline algorithm. An online algorithm A for OLTP is *c-competitive*, if there exists a constant c such that for any request sequence σ:

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

3. Related Work

The model of online computation using release dates for the requests was introduced in [AF+95, AF+94]. There, the authors studied a similar problem which they called the *online traveling salesman problem* (OLTSP): A server moves at constant unit speed through a metric space and serves requests that occur at points of the space. The main difference between the OLTSP and the OLTP studied in this paper are the following:

- In the OLTSP the server can change its mind any time, recompute a new schedule and, if necessary, change its direction right on the spot. In the OLTP this is not possible. Once the server has picked up an object to be transported from a to b it is not allowed to drop that object at any other place than b.
- The distances in the metric space of the OLTSP are assumed to be symmetric. Thus, if starting and ending in the origin, a set $\sigma = \{r_1, \dots, r_m\}$ of requests can be served in the order $r_1, \dots, r_m$ or $r_m, \dots, r_1$ at the same cost. In the OLTP changing the "direction" of service could increase the cost by a factor of two.

In [AF+95, AF+94] the authors show a lower bound of $\frac{9+\sqrt{17}}{8} \approx 1.64$ for the competitive ratio of any deterministic online algorithm for the OLTSP. This lower bound is established for the special case that the metric space is the real line. In Section 4 we show a slightly better lower bound of $1+\sqrt{2}/2 \approx 1.70$ for the OLTP on the real line.

The authors in [AF+95, AF+94] also present a $7/4$-competitive algorithm for the real line and a 2-competitive algorithm that works in an arbitrary (symmetric) metric space.

Recently, in an independent effort Feuerstein and Stougie [FS98] analyzed the algorithm Ignore(which they call DLT) and established the same competitive ratio as in this paper. To the best of our knowledge, there have been no proofs of the competitive ratios of our other algorithms in literature so far.

4. Lower Bounds

In this section we address the question how well an online algorithm can perform compared to the optimal offline adversary.

Theorem 4.1. *No deterministic algorithm for* OLTP *can achieve a competitive ratio* $c < 5/3$.

Proof. The underlying graph $G = (V, E)$ for the instance of OLTP consists of a path of 5 vertices $v_0, \dots, v_4$ with the origin being the "leftmost" vertex v_0 at distance $D = 4$ from the right end. All edge-weights are equal to one, so that $d(o, v_i) = i$.

Suppose that A is a deterministic online algorithm with competitive ratio c. We can assume that $c \leq 5/3$, since otherwise there is nothing left to be proved.

At time $t = 0$, the algorithm A is faced with two requests $r_1 = (0, o, v_2)$ and $r_2 = (0, v_2, o)$. Thus $C_{\mathsf{OPT}}(r_1, r_2) = 4$ and the server operated by A must start serving request r_2 at some time $2 \leq T \leq 4c-2$. Since $c \leq 5/3$, we have $2 \leq T \leq 4c-2 \leq 4\frac{2}{3}$.
Case 1: $2 \leq T \leq 3$.

At time T the adversary issues another request $r_3 = (T, v_3, v_2)$. Thus, the online server can not finish before time $T + 8 \geq 10$. On the other hand, the offline server first handles r_1, then continues to move to v_3 which it reaches no earlier than the time when r_3 becomes known. Hence, the optimal offline server incurs a total cost of 6, which gives us that

$$\frac{C_{\mathsf{A}}(r_1, r_2, r_3)}{C_{\mathsf{OPT}}(r_1, r_2, r_3} \geq \frac{5}{3}.$$

Case 2: $3 < T \leq 4c-2$.

In this case, at time T the adversary issues another request $r_3 = (T, v_{\lfloor T \rfloor}, v_2)$. Notice that since $T > 3$ and $\lfloor T \rfloor \leq 4$ we have that $v_{\lfloor T \rfloor} \in \{v_3, v_4\}$.

Let $\lfloor T \rfloor = T + \varepsilon$ for some $-1 \leq \varepsilon \leq 0$. The online algorithm will need total time at least $T + 2 + 2\lfloor T \rfloor = 3T + 2 + 2\varepsilon$. The offline server first serves request r_1 and then moves to vertex $v_{\lfloor T \rfloor}$ where it sits until time T. It then serves r_3 and, finally, r_2 at a total cost of $C_{\mathsf{OPT}}(r_1, r_2, r_3) = T + \lfloor T \rfloor = 2T + \varepsilon$.

Thus in the second case the ratio between the time needed by A and the optimal offline algorithm is

$$\frac{C_{\mathsf{A}}(r_1, r_2, r_3)}{C_{\mathsf{OPT}}(r_1, r_2, r_3)} \geq \frac{3T + 2 + 2\varepsilon}{2T + \varepsilon} \geq \frac{3(4c-2) + 2 + 2\varepsilon}{2(4c-2) + \varepsilon} = \frac{12c - 4 + 2\varepsilon}{8c - 4 + \varepsilon}$$

It is easy to check that the expression given above is increasing in ε. Thus we have that the competitive ratio c is bounded from below by the value of the above expression for $\varepsilon = -1$:

$$c \geq \frac{12c - 6}{8c - 5}. \tag{1}$$

The smallest value $c \geq 1$ satisfying Equation (1) is $c = \frac{17+\sqrt{97}}{16} \approx 1.678 > \frac{5}{3}$, which contradicts the assumption that A is c-competitive with $c < 5/3$. □

It should be pointed out that the lower bound above is achieved on a path where the ratio of the diameter and the minimum distance between any two vertices is bounded by 4. For the case when the underlying metric space for OLTP is the real line we can establish a slightly better lower bound by basically the same construction as in the proof of Theorem 4.1:

Theorem 4.2. *For the* OLTP *on the real line, no deterministic algorithm can achieve a competitive ratio* $c < 1 + \sqrt{2}/2 \approx 1.7071068$. □

5. Two Simple Strategies

In this section we present and analyze two very natural online-strategies for for OLTP. We also study another somewhat less natural strategy which shows that leaving the server idle for some time can also lead to a competitive algorithm.

Strategy Replan: As soon as a new request arrives, the server completes the current carrying move (if it is performing one), then the server stops and does a replan: it computes a new shortest schedule which starts at the current position of the server, takes care of all yet unserved requests and then either stops or returns to the origin, depending on the problem specification, i.e., whether the server should move along *open* or *closed* schedules.

Strategy Ignore: The server remains idle until the first request becomes known. It then serves the first request immediately. All requests that arrive during the service of the first request are temporarily ignored. After the first request has been served, the server computes a shortest schedule for all unserved requests and follows this schedule. Again, all new requests that arrive during the time that the server is following the schedule are temporarily ignored. A schedule for the ignored requests is computed as soon as the server has completed its current schedule. The algorithm keeps on following schedules and temporarily ignoring requests this way.

For a set σ of requests and a point x let $L^*(t,x,\sigma)$ denote the length of a shortest schedule (i.e., the time difference between its completion time and the start time t) which starts in x at time t, serves all requests from σ and ends in the origin. Clearly for $t' \geq t$ we have that $L^*(t',x,\sigma) \leq L^*(t,x,\sigma)$. Moreover, $C_{\mathsf{OPT}}(\sigma) = L^*(0,o,R)$ and thus $C_{\mathsf{OPT}}(\sigma) \geq L^*(t,o,\sigma)$ for any time $t \geq 0$.

Since the optimum offline server OPT can not serve the last request $r_m = (t_m, a_m, b_m)$ from σ before this request is available we get that

$$C_{\mathsf{OPT}}(\sigma) \geq \max\{L^*(t,o,\sigma), t_m + d(a_m,b_m) + d(b_m,o)\} \quad \text{for any } t \geq 0. \tag{2}$$

Lemma 5.1. *Let $\sigma = \{r_1,\dots,r_m\}$ be a set of requests. Then for any $t \geq t_m$ and any request $r_i = (t_i,a_i,b_i)$ from σ*

$$L^*(t,b_i,\sigma \setminus r_i) \leq L^*(t,o,\sigma) - d(a_i,b_i) + d(a_i,o).$$

Proof. Consider an optimum schedule S^* which starts at the origin o at time t, serves all requests in σ and has length $L^*(t,o,\sigma)$. It suffices to construct another schedule S which starts in b_i no earlier than time t serves all requests in $\sigma \setminus r_i$ and has length at most $L^*(t,o,\sigma) - d(a_i,b_i) + d(a_i,o)$.

Let S^* serve the requests in the order $r_{j_1},\dots,r_{j_m}$ such that $r_i = r_{j_k}$. Notice that if we start in b at time t and serve the requests in the order

$$r_{j_{k+1}},\dots,r_{j_m},r_{j_1},\dots,r_{j_{k-1}}$$

and then moving back to the origin yields a schedule S with the desired properties. □

We are now ready to prove the result about the performance of Replan:

Theorem 5.2. *Algorithm Replan is 5/2-competitive.*

Proof. Let $\sigma = \{r_1,\dots,r_m\}$ be any set of requests. We distinguish between two cases depending on the current load of the Replan-server at the time t_m (the last request becomes known).

If the server is currently empty it recomputes an optimal schedule which starts at its current position, denoted by $s(t_m)$, serves all unserved requests, and returns to the origin. This schedule has length at most $L^*(t_m,s(t_m),\sigma) \leq d(o,s(t_m)) + L^*(t_m,o,\sigma)$. Thus,

$$C_{\mathsf{Replan}}(\sigma) \leq t_m + d(o,s(t_m)) + L^*(t_m,o,\sigma) \overset{(2)}{\leq} t_m + d(o,s(t_m)) + C_{\mathsf{OPT}}(\sigma). \tag{3}$$

New now consider the second case, when the server is currently serving a request $r = (t, a, b)$. The time needed to complete the move is $d(s(t_m), b)$. Then a shortest schedule starting at b serving all unserved requests is computed which has length at most $L^*(t_m, b, \sigma \setminus r)$. Thus in the second case

$$\begin{aligned}
C_{\mathsf{Replan}}(\sigma) &\leq t_m + d(s(t_m), b) + L^*(t_m, b, \sigma \setminus r) \\
&\leq t_m + d(s(t_m), b) + L^*(t_m, o, \sigma) - d(a, b) + d(a, o) \quad \text{(by Lemma 5.1)} \\
&\leq t_m + C_{\mathsf{OPT}}(\sigma) - d(a, b) + \underbrace{d(s(t_m), b) + d(a, s(t_m))}_{=d(a,b)} + d(s(t_m), o) \\
&= t_m + d(o, s(t_m)) + C_{\mathsf{OPT}}(\sigma).
\end{aligned}$$

This means that inequality (3) holds in both cases. Since the Replan server has traveled to position $s(t_m)$ at time t_m, there must be a request $r_j = (t_j, a_j, b_j)$ in σ where either $d(o, a_i) \geq d(o, s(t_m))$ or $d(o, b_i) \geq d(o, s(t_m))$. But this means that the optimal offline server will have to travel at least twice the distance $d(o, s(t_m))$ during its schedule. Thus, $d(o, s(t_m)) \leq C_{\mathsf{OPT}}(\sigma)/2$ and using this result together with (3) we get that the total time the Replan server needs is no more than $5/2$ times that of the offline server. □

We are now going to analyze the competitiveness of the second simple strategy Ignore.

Theorem 5.3. *Algorithm Ignore is $5/2$-competitive.*

Proof. Consider again the moment t_m in time when the last request r_m becomes known. If the Ignore server is currently idle at the origin o, then its completes its last schedule no later than $t_m + d(o, a_m) + d(a_m, b_m) + d(b_m, o) \leq C_{\mathsf{OPT}}(\sigma) + d(o, a_m) \leq 3/2 \cdot C_{\mathsf{OPT}}(\sigma)$. Here, we have used again the fact that the optimum offline server will have to travel at least twice the distance $d(o, a_m)$ during its service.

It remains the case that at time t_m the Ignore server is currently working on a schedule S for a subset σ_S of the requests. Let t_S denote the starting time of this schedule. Thus, the Ignore-server will complete S at time $t_S + L^*(t_S, o, \sigma_S)$. Denote by $\sigma_{\geq t_S}$ the set of requests presented after time t_S. Notice that $\sigma_{\geq t_S}$ is exactly the set of requests that are served by Ignore in its last schedule. The Ignore-server will complete its total service no later than time $t_S + L^*(t_S, o, \sigma_S) + L^*(t_m, o, \sigma_{\geq t_S})$.

Let $r_f \in \sigma_{\geq t_S}$ be the first request from $\sigma_{\geq t_S}$ served by OPT. Thus

$$C_{\mathsf{OPT}}(\sigma) \geq t_f + L^*(t_f, a_f, \sigma_{\geq t_S}) \geq t_S + L^*(t_m, a_f, \sigma_{\geq t_S}). \tag{4}$$

Now $L^*(t_m, o, \sigma_{\geq t_S}) \leq d(o, a_f) + L^*(t_m, a_f, \sigma_{\geq t_S})$ and $L^*(t_S, o, \sigma_S) \leq C_{\mathsf{OPT}}(\sigma)$. This gives us that

$$\begin{aligned}
C_{\mathsf{Ignore}}(\sigma) &\leq t_S + C_{\mathsf{OPT}}(\sigma) + d(o, a_f) + L^*(t_m, a_f, \sigma_{\geq t_S}) \\
&\overset{(4)}{\leq} 2C_{\mathsf{OPT}}(\sigma) + d(o, a_f) \\
&\leq 5/2 \cdot C_{\mathsf{OPT}}(\sigma).
\end{aligned}$$

This completes the proof. □

6. The Sleep Strategy

Strategy Sleep is another very simple strategy that surprisingly leads to a competitive algorithm.

Strategy Sleep: The algorithm has a fixed "waiting scaling" parameter $\theta > 1$. The algorithm also records a "base time" T which equals zero initially. From time to time the algorithm consults its "work-or-sleep" routine: this subroutine computes a shortest schedule for all unserved requests. If this schedule can be completed before time θT the subroutine returns (S, work), otherwise it returns (S, sleep).

The server of algorithm Sleep can be in four states:

idle: In this case the server has served all known requests, is sitting in the origin and waiting for new requests to occur.

sleeping: In this case the server knows of some unserved requests but also knows that they take too long to serve (what "too long" means will be formalized in the algorithm below).

working: In this state the algorithm (or rather the server operated by it) is following a computed schedule.

We now formalize the behavior of the algorithm by specifying how it reacts in each of the four states.

- If the algorithm is idle and a new request arrives, it updates the base time T to the release time of the new request (which equals the current time). It then calls "work-or-sleep". If the result is (S, work), the algorithm resets T to the completion time of S and enters the working state.
  If the result of "work-or-sleep" is (S, sleep), then the algorithm enters the sleeping state.
- In the sleeping state the algorithm resets its base time to $T' = \theta T$ and simply does nothing (or sleeps) until time T'. At time T' the algorithm reconsults its "work-or-sleep" subroutine. This process is continued until the server eventually enters the working state (since the number of requests is finite).
- In the working state, i.e, while the server is following a schedule all new requests are ignored. As soon as the current schedule is completed the server either enters the idle-state (if there are no unserved requests) or it consults its "work-or-sleep" subroutine which determines the next state.

Theorem 6.1. *Algorithm Sleep is* $\max\left\{\frac{5}{2}, 2 + \frac{1}{\theta-1}\right\}$*-competitive.*

Proof. Let $\sigma_{=t_m}$ be the set of requests released at time t_m, i.e., the point in time when the last requests becomes known. We distinguish between different cases depending on the state of the Sleep-server at time t_m:

Case 1: The server is idle.

In this case the algorithm computes a shortest schedule for the requests in $\sigma_{=t_m}$ and resets its base time to $T = t_m$. The Sleep-server will start its work at time $T\theta^i$, where i is the smallest nonnegative integer such that $T\theta^i + L^*(T\theta^i, o, \sigma_{=t_m}) \leq T\theta^{i+1}$. Notice that for all i we have $L^*(T\theta^i, o, \sigma_{=t_m}) = L^*(t_m, o, \sigma_{=t_m}) \leq C_{\mathsf{OPT}}(\sigma)$.

In other words, the server starts its work at time $T\theta^i$ such that $i \geq 0$ is the smallest nonnegative integer such that

$$L^*(t_m, o, \sigma_{=t_m}) \leq T\theta^i(\theta - 1). \tag{5}$$

The server will complete its work at time $T\theta^i + L^*(t_m, o, \sigma_{=t_m})$.

If $i = 0$, then $C_{\mathsf{Sleep}}(\sigma) = t_m + L^*(t_m, o, \sigma_{=t_m}) \leq 2 \cdot C_{\mathsf{OPT}}(\sigma)$. If $i > 0$, then by the minimality of i we have that Equation (5) is false for $i-1$ and we get that

$$C_{\mathsf{OPT}}(\sigma) \geq L^*(t_m, o, \sigma_{=t_m}) > T\theta^{i-1}(\theta - 1). \tag{6}$$

On the other hand

$$\begin{aligned} C_{\mathsf{Sleep}}(\sigma) &\leq T\theta^i + L^*(t_m, o, \sigma_{=t_m}) \\ &\overset{(6)}{<} \frac{\theta}{\theta - 1} C_{\mathsf{OPT}}(\sigma) + L^*(t_m, o, \sigma_{=t_m}) \\ &\leq \left(1 + \frac{\theta}{\theta - 1}\right) C_{\mathsf{OPT}}(\sigma) \\ &= \left(2 + \frac{1}{\theta - 1}\right) C_{\mathsf{OPT}}(\sigma). \end{aligned}$$

Case 2: The server is sleeping.

Let $T\theta^p$ for some $p \geq 0$ be the time when the algorithm went to sleep the last time. At time $T\theta^{p+1}$ the algorithm will reconsult its "work-or-sleep" subroutine. We have that $T\theta^p \leq t_m \leq T\theta^{p+1}$.

The server will start its last schedule at time $T\theta^{p+1+i}$, where i is the smallest integer such that $T\theta^{p+1+i} + L^*(t_m, o, \sigma') \leq \delta\theta^{p+i+2}$, where σ' denotes the set of yet unserved requests. By essentially the same arguments as in Case 1 we get that

$$C_{\mathsf{Sleep}}(\sigma) \leq \left(2 + \frac{1}{\theta - 1}\right) C_{\mathsf{OPT}}(\sigma).$$

Case 3: The algorithm is working.

If after completion of the current schedule S the server enters the sleep state then the arguments given above establish that the completion time does not exceed $\left(2 + \frac{1}{\theta-1}\right) C_{\mathsf{OPT}}(\sigma)$.

The remaining case is that the Sleep-server starts its final schedule immediately after having completed S. Thus, from the time t_S where the server started S, the Sleep algorithm behaves exactly like the Ignore strategy and the arguments given in the proof of Theorem 5.3 show that in this case $C_{\mathsf{Sleep}}(\sigma) \leq 5/2 \cdot C_{\mathsf{OPT}}(\sigma)$. This completes the proof. □

We note that the second factor $1 + \frac{\theta}{\theta-1} = 2 + \frac{1}{\theta-1}$ is at most $5/2$ provided that $\theta \geq 3$. We thus obtain the following corollary.

Corollary 6.2. *If $\theta \geq 3$, then algorithm Sleep is $5/2$-competitive.* □

7. Extension to Open Schedules

In this section we show how to extend our results to the case of "open schedules" where the server is not required to return to the origin at the end of its service.

7.1. Lower Bound.

We first establish a lower bound on the competitive ratio of any deterministic algorithm for OLTP with open schedules.

Theorem 7.1. *For open schedules no deterministic algorithm for* OLTP *can achieve a competitive ratio $c < 2 - 4/D$, where $D := \max_{v \in V} d(o, v)$ is the maximum distance of any node in the graph to the origin vertex o.*

Proof. The underlying graph $G = (V, E)$ for the instance of OLTP consists of a path of $2n+1$ vertices with the origin being the middle vertex at distance of n from each end. More formally, we have that $V = \{ v_i : i = -n, \ldots, n \}$ with $o := v_0$ and $E = \{ (v_i, v_{i+1}) : i = -n, \ldots, n-1 \}$. All edge-weights are equal to 1.

Assume that A ist an Algorithm with competitive ratio $c < 2 - 4/D = 2 - 4/n$. The first request given is $r_1 = (t_1 = 0, v_0, v_{-1})$. We now claim that at time $t_2 = n-1$ the online-server can not be strictly to the right of vertex v_1. In fact, if the server were to the right, say at distance $\delta > 0$ to the right of vertex v_1, then we could add the request $r_2 = (n-1, v_{-(n-1)}, v_{-n})$. The online server would then need time at least $1 + \delta + n$ to serve the request. This would result in a total time of $2n + \delta$. On the other hand, the offline server could serve r_1 starting at time $t_1 = 0$ and then continue to move to the left until it reaches vertex $v_{-(n-1)}$ at time $n-1$ ready to serve the new request r_2. Thus, the offline server needs time n and thus $C_{\mathsf{A}}(r_1, r_2)/C_{\mathsf{OPT}}(r_1, r_2) = (2n + \delta)/n > 2$ which means that A would not even be 2-competitive.

We have seen that at time $t_2 = n-1$ the online server is to the left of vertex v_1. We now add the request $r_3 = (n-1, v_{n-3}, v_{n-2})$. The total time needed by the online server is then at least $n - 1 + (n-2) = 2n - 4$.

On the other hand, the offline server serves the sequence r_1, r_3 by handling r_1 at time t_1 and then immediately moving to vertex v_{n-2} which it reaches at time $n-1$ ready to serve r_3. Thus we have that $C_{\mathsf{OPT}}(r_1, r_3) = n$ and $C_{\mathsf{A}}(r_1, r_3)/C_{\mathsf{OPT}}(r_1, r_3) = 2 - 4/n > c$. This contradicts the assumption that A is c-competitive with $\alpha < 2 - 4/D$. □

7.2. **Competitive Ratios of Replan and Ignore.** We now show how to modify the proof of Theorem 5.2 to obtain a result about the competitiveness of Replan for open schedules.

For a set σ of requests and a point x let $\tilde{L}^*(t, x, \sigma)$ denote the length of a shortest schedule (i.e., the time difference between its completion time and the start time t) which starts in x at time t, serves all requests from σ. The difference to $L^*(t, x, \sigma)$ defined in Section 5 is that we do not require the schedule to end at the origin.

For $t' \geq t$ we have that $\tilde{L}^*(t', x, \sigma) \leq \tilde{L}^*(t, x, \sigma)$. Moreover, $C_{\mathsf{OPT}}(\sigma) = \tilde{L}^*(0, o, \sigma)$ and thus $C_{\mathsf{OPT}}(\sigma) \geq \tilde{L}^*(t, o, \sigma)$ for any time $t \geq 0$. Since the optimum offline server OPT can not serve the last request $r_m = (t_m, a_m, b_m)$ from σ before this request is available we get that

$$C_{\mathsf{OPT}}(\sigma) \geq \max\{L^*(t, o, \sigma), t_m + d(a_m, b_m)\} \quad \text{for any } t \geq 0. \tag{7}$$

The following lemma can be proved similarly to Lemma 5.1:

Lemma 7.2. *Let $\sigma = \{r_1, \ldots, r_m\}$ be a set of requests. Then for any $t \geq t_m$ and any request $r_i = (t_i, a_i, b_i)$ from σ*

$$\tilde{L}^*(t, b_i, \sigma \setminus r_i) \leq \tilde{L}^*(t, o, \sigma) - d(a_i, b_i) + d(b, o),$$

where b is the endpoint of a path with length $\tilde{L}^*(t, o, \sigma)$. In particular

$$\tilde{L}^*(t, b_i, \sigma \setminus r_i) \leq 2\tilde{L}^*(t, o, \sigma) - d(a_i, b_i).$$

We are now ready to establish the analogon of Theorem 5.2 for open schedules:

Theorem 7.3. *In the case of open schedules Algorithm Replan is 3-competitive.* □

Proof. If at the time t_m when the last request $r_m = (t_m, a_m, b_m)$ from σ is issued, the Replan server does not perform a carrying move, then the total time needed by Replan is no more than

$$t_m + \tilde{L}^*(t_m, s(t_m), \sigma) \le t_m + d(s(t_m), o) + \tilde{L}^*(t_m, o, \sigma) \le d(o, s(t_m)) + 2C_{\mathsf{OPT}}(\sigma).$$

The distance $d(o, s(t_m))$ can be bounded from above by $C_{\mathsf{OPT}}(\sigma)$ (instead of $C_{\mathsf{OPT}}(\sigma)/2$ as for closed schedules) and thus Replan needs time at most 3 times the optimum in this case.

If at time t_m Replan performs a carrying move from a to b, it will finish its move at time $t_m + d(s(t_m), b)$. We thus have

$$\begin{aligned} C_{\mathsf{Replan}}(\sigma) &\le t_m + d(s(t_m), b) + \tilde{L}^*(t_m, b, \sigma) \\ &\le t_m + d(s(t_m), b) + 2\tilde{L}^*(t_m, o, \sigma) - d(a, b) && \text{(by Lemma 7.2)} \\ &\le t_m + 2\tilde{L}^*(t_m, o, \sigma) \\ &\le 3 \cdot C_{\mathsf{OPT}}(\sigma). \end{aligned}$$

This completes the proof. □

Theorem 7.4. *In the case of open schedules, the competitive ratio of Algorithm Ignore is* 4.

Proof. The only interesting case is that at the time t_m when the last request becomes known the Ignore server is currently working on a schedule S. Suppose that S was started at time t_S and has starting point x and ends at point y. Then the schedule will be completed no later than time $t_S + \tilde{L}^*(t_S, x, \sigma_S)$, where σ_S denotes the subset of requests served in the current schedule S. The Ignore server will complete its total work no later than time

$$t_S + \tilde{L}^*(t_S, x, \sigma_S) + \tilde{L}^*(t_m, y, \sigma_{\ge t_S}),$$

where $\sigma_{\ge t_S}$ denotes the set of requests presented after time t_S.

Let r_f be the first request from the set $\sigma_{\ge t_S}$ of ignored requests served by OPT. Then

$$C_{\mathsf{OPT}}(\sigma) \ge t_f + \tilde{L}^*(t_S, a_f, \sigma_{\ge t_S}) \ge t_S + \tilde{L}^*(t_S, a_f, \sigma_{\ge t_S}).$$

Thus, we have that

$$\begin{aligned} C_{\mathsf{Ignore}}(\sigma) &\le \tilde{L}^*(t_S, x, \sigma_S) + d(y, a_f) + t_S + \tilde{L}^*(t_m, a_f, \sigma_{\ge t_S}) \\ &\le \tilde{L}^*(t_S, x, \sigma_S) + d(y, a_f) + C_{\mathsf{OPT}}(\sigma) \\ &\le 2C_{\mathsf{OPT}}(\sigma) + d(x, o) + d(y, a_f). \end{aligned}$$

It is easy to see that both values $d(x, o)$ and $d(y, a_f)$ are bounded from above by $C_{\mathsf{OPT}}(\sigma)$, and so the theorem follows. □

8. Concluding Remarks

The competitive ratios of our three algorithms are summarized in Table 1. Examples show that all of them are tight.

Algorithm	Replan	Ignore	Sleep
Closed Schedules	$\frac{5}{2}$	$\frac{5}{2}$	$\max\left\{\frac{5}{2}, 2+\frac{1}{\theta-1}\right\}$
Closed Schedules	3	4	—

TABLE 1. Competitive Ratios of algorithms in this paper.

REFERENCES

[AF+94] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo, *Serving request with on-line routing*, Proceedings of the 4th Scandinavian Workshop on Algorithm Theory (SWAT'94),, Lecture Notes in Computer Science, vol. 824, July 1994, pp. 37–48.

[AF+95] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo, *Competitive algorithms for the traveling salesman*, Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95), Lecture Notes in Computer Science, vol. 955, August 1995, pp. 206–217.

[AG+98] N. Ascheuer, M. Grötschel, S. O. Krumke, and J. Rambau, *Combinatorial online optimization*, Proceedings of the International Conference of Operations Research Zürich (OR'98), Springer, 1998, (to appear).

[AK88] M. J. Atallah and S. R. Kosaraju, *Efficient solutions to some transportation problems with applications to minimizing robot arm travel*, SIAM Journal on Computing **17** (1988), no. 5, 849–869.

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

[ET76] K. P. Eswaran and R. E. Tarjan, *Augmentation problems*, SIAM Journal on Computing **5** (1976), 653–665.

[Fre93] G. N. Frederickson, *A note on the complexity of a simple transportation problem*, SIAM Journal on Computing **22** (1993), no. 1, 57–61.

[FS98] E. Feuerstein and L. Stougie, *On-line single server dial-a-ride problems*, Manuscript, submitted for publication, 1998.

[FW98] A. Fiat and G. J. Woeginger (eds.), *Online algorithms: The state of the art*, Lecture Notes in Computer Science, vol. 1442, Springer, 1998.

KONRAD-ZUSE-ZENTRUM FÜR INFORMATIONSTECHNIK BERLIN, DEPARTMENT OPTIMIZATION, TAKUSTR. 7, 14195 BERLIN-DAHLEM, GERMANY.

E-mail address: {ascheuer, krumke, rambau}@zib.de