On Minimizing the Maximum Flow Time in the Online Dial-a-Ride Problem

Sven O. Krumke¹, Willem E. de Paepe², Diana Poensgen³, Maarten Lipmann², Alberto Marchetti-Spaccamela⁴, and Leen Stougie³

¹ University of Kaiserslautern, Department of Mathematics, P.O.Box 3049, Paul-Ehrlich-Str. 14, 67653 Kaiserslautern, Germany. E-mail: krumke@mathematik.uni-kl.de

² Combinatorial Optimization Group, Technical University of Eindhoven, P. O. Box 513, 5600MB Eindhoven, The Netherlands. Email: {leen,w.e.d.paepe}@win.tue.nl

³ Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization, Takustr. 7, 14195 Berlin, Germany. E-mail: poensgen@zib.de

⁴ Dipartimento di Informatica e Sistemistica, Università di Roma “La Sapienza”, Via Salaria 113, 00198 Rome, Italy. E-mail: alberto@dis.uniroma1.it

Abstract. In the online dial-a-ride problem (OLDARP), objects must be transported by a server between points in a metric space. Transportation requests (“rides”) arrive online, specifying the objects to be transported and the corresponding source and destination.

We investigate the OLDARP for the objective of minimizing the maximum flow time. It has been well known that there can be no strictly competitive online algorithm for this objective and no competitive algorithm at all on unbounded metric spaces. However, the question whether on metric spaces with bounded diameter there are competitive algorithms if one allows an additive constant in the definition competitive ratio, had been open for quite a while. We provide a negative answer to this question already on the uniform metric space with three points. Our negative result is complemented by a strictly 2-competitive algorithm for the Online Traveling Salesman Problem on the uniform metric space, a special case of the problem.

1 Introduction

In the Dial-a-Ride Problem (DARP), a server of unit capacity has to transport objects through a given metric space. The server starts at a designated point of the metric space, its *origin*. Once the server has picked up an object, it can only drop it at its destination. A special case of the DARP is the Traveling Salesman Problem (TSP) in which the server merely has to visit points in that metric space.

In the online version of the problem, *online Dial-a-Ride problem* requests (also called *rides*) arise while the server is already moving. Each request r_j specifies a release time $t_j \geq 0$, a source u_j , and a destination v_j . An online algorithm only learns about r_j at its release time. The objective we consider is

to minimize the *maximum flow time*. If request r_j is served at time t , its flow time is $t - t_j$. We abbreviate the resulting problem by $F_{\max}$ -OLDARP. The maximum flow time can be identified with the maximal dissatisfaction of customers who are waiting to be transported or to receive a desired good. Natural applications include elevator control, delivery services, and craftsmen on duty.

Given a sequence $\sigma = r_1, \dots, r_m$ of requests, we denote by $\text{ALG}(\sigma)$ the maximum flow time in the solution provided by algorithm ALG . We evaluate the quality of online algorithms by competitive analysis [4]: an online algorithm ALG is *c-competitive*, if there exists $b \geq 0$ such that

$$\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + b \quad (1)$$

holds for all request sequences σ , where OPT denotes an optimal offline algorithm, which has complete knowledge about the input at time 0. If (1) holds with $b = 0$, then ALG is termed *strictly c-competitive*. Competitive analysis can be imagined as a game between an online player and a malicious *adversary* who tries select a worst-case request sequence which maximizes the ratio between the online and the offline cost.

Online Dial-a-Ride problems have been previously investigated with various objective functions and for different metric spaces [1–3, 5, 6, 9–11]. It is well known that for the $F_{\max}$ -OLDARP in general metric spaces no strictly competitive algorithms can exist, see e.g. [7, 8, 11]. For the special case of the $F_{\max}$ -OLTSP, where source and destination for each ride coincide ($u_j = v_j$ for all j), a restriction on the adversary allows for a strictly competitive algorithm on the real line [11].

In this paper we study we study the $F_{\max}$ -OLDARP and $F_{\max}$ -OLTSP on the *uniform metric space* with n points, where any two distinct points have distance one. This can be envisioned as a complete graph K_n with unit length edges. Only the n points can occur as source or destination. We allow servers to move continuously at unit speed along the edges.

Observe that with $b = n$ in (1), a simple online algorithm that visits the n nodes of the uniform metric space in a round-robin manner is 1-competitive for the $F_{\max}$ -OLTSP. For the $F_{\max}$ -OLDARP, however, it had been an open question whether allowing an additive constant $b > 0$ allows to prove (positive) competitiveness results. We resolve this question.

Contribution and Paper Outline

In this paper we show that on the uniform metric space with $n = 3$ points neither an arbitrary additive constant b nor restricting the adversary to be fair in the sense of [3, 11] allows for competitive algorithms. On the uniform metric space an adversary is *fair*, if at any moment $t > 0$, her server is located between two points, each of which is the origin or has occurred as source or destination of a request with release time at most t (this definition extends the notion of fairness given in [3] for the real line).

We also investigate the $F_{\max}$ -OLTSP on the uniform metric space against the fair adversary and prove that a simple first-come-first-serve strategy is strictly 2-competitive, which we prove to be best possible for online algorithms.

2 A Negative Result for the $F_{\max}$ -OLDarp

In this section we shall prove the following theorem:

Theorem 1. *For the $F_{\max}$ -OLDARP on the uniform metric space, no deterministic online algorithm can be competitive even against a fair adversary.*

Proof. Assume for the sake of a contradiction that ALG is a c -competitive deterministic algorithm with additive constant b as in (1). W. l. o. g., we may assume that c is integral. In the sequel we will also assume that at integral points in time, ALG's server is located in one of the nodes. Note that any online algorithm can be transformed into another online algorithm with this property at the cost of an additive constant of one. So, this assumption is without loss of generality for the proof.

We show that for any $k \in \mathbb{N}$ we can construct a finite sequence $\sigma = \sigma(k)$ such that $\text{OPT}(\sigma) \leq 3$, whereas $\text{ALG}(\sigma) \geq k$. This contradicts the fact that ALG is c -competitive and also rules out any additive constant (one just has to choose k appropriately depending on c and b).

Our construction just uses the subgraph of K_n induced by the origin and two additional points. The resulting three points and edges are denoted by x, y, z and X, Y, Z , see Figure 1. Our sequence has the following properties: (i) Requests are only given at integer times, (ii) at any time unit, at most one request is given, and, (iii) in any two consecutive time units, at least one request is given.

A crucial ingredient for our construction is that of an *empty move*, in which a server moves from a node u to some other node v without serving a request (t, u, v) . The main idea is to enforce an empty move for the online algorithm which can be avoided by the adversary. This way, work piles up for ALG, while the number of unserved requests for the adversary remains bounded by a constant (namely, three).

Fig. 1. Metric space and notation for the lower bound construction.

In the sequel we describe a request by the corresponding edge and a direction, which depends on the direction of the previous request for that edge. Two requests for the same edge with different directions are called *opposed*. We use $\sigma_{\leq t}$ to denote the requests in σ with release time at most t .

The sequence is constructed in phases. In each phase, the number of unserved requests for the online algorithm increases by one. More precisely, the k th phase starts at time T_k when the following property P_k is satisfied:

Property P_k : There exists $i \in \{1, 2, 3\}$ such that:

- (i) ALG has k unserved requests for e_i , none for all the other edges, and is located in one of e_i 's end points, $\lceil k/2 \rceil$ requests directed away from ALG's position, and $\lfloor k/2 \rfloor$ requests directed the other way.
- (ii) $\text{ADV}(\sigma_{\leq T_k}) \leq 3$, and ADV's server is located in one of e_i 's end points with exactly two opposed requests for e_i pending, one of which has been released at T_k , the other one at time $T_k - 1$.

Figure 2 displays the scenario described by Property P_k .

Fig. 2. Property P_k .

Claim 1. Assume that P_k holds at time T_k . Then, the adversary can release further requests such that, at some time $T_{k+1} \geq T_k$ Property P_{k+1} holds.

Proof. Proof of Claim 1 Assume that Property P_k holds at time T_k with $i = 2$ and that the adversary is positioned at node 1 (the other cases are symmetric). We have to distinguish two cases depending on the parity of k .

Case 1: If ALG served all pending e_2 -requests in a row, it would finish in node 3 (left part of Figure 2 if k is even; right part of the figure if k is odd).

The sequence continues with a request for e_3 (direction arbitrary) at time $T_k + 1$. The adversary then releases one request for e_3 at each integer time (directions alternating), until at some time T ALG has served the last pending e_2 -request and is located in one of e_3 's end points, either node 1 or node 2. Observe that such a time must exist since otherwise ALG can not be competitive at all). Let $T' \leq T$ be the earliest time when ALG has served all pending e_2 -requests. At time T' , ALG is either in node 1 or in node 3. In the former case, ALG must have made an empty move: it must have either moved along e_2 without serving a request or it has moved around the triangle and made an empty move

along e_1 (this uses the assumption of Case 1). We proceed to show that Property P_{k+1} holds at time $T_{k+1} := T$.

Starting at time T_k , ADV serves the two pending requests for e_2 and then serves one request for e_3 in each time unit. Clearly, at time T , she is located either in 1 or 2 and has two pending requests for e_3 , one released at time $T - 1$ and the other one at T . This ensures part (ii) of Property P_{k+1} .

To prove (i), we compute how many unserved e_3 -requests have piled up for ALG by time T . ALG has at least k requests for e_2 pending at time T_k . Since it can serve at most one request per time unit and must serve k pending requests on e_2 , at time T' at least k requests for e_3 have piled up (no matter whether some of the e_3 -requests have been served before time T').

If $p^{\text{ALG}}(T') = 1$, then $T = T'$ and as we have seen above, ALG must have made an empty move which results in an extra unserved request for e_3 . If $p^{\text{ALG}}(T') = 3$, ALG needs at least one empty move in order to reach node 1 or 2 at time $T \geq T' + 1$. In any case, at time T ALG has at least $k + 1$ pending requests for e_3 . Hence, part (i) of Property P_k follows.

Case 2: If ALG served all pending e_2 -requests in a row, it would finish in node 1.

In this case the sequence is continued until the prerequisites of Case 1 are met and we can continue as described above (with a suitable cyclic exchange of indices)

At time $T_k + 1$, no request is given. Starting at time $T_k + 2$, the adversary gives one request for edge e_1 in each time unit (directions alternating) until time τ , the earliest time at which ALG has finished serving the last among the pending e_2 -requests and is located in one of e_1 's end nodes, either 2 or 3. Let $\tau' \leq \tau$ be the earliest time at which ALG has finished serving the pending e_2 -requests. Again, the existence of $\tau \in \mathbb{N}$ is ensured by the assumption that ALG is competitive.

ALG can serve at most one of the k pending e_2 -requests in each time unit. Since one new request for e_1 is given in each time unit except for $T_k + 1$, ALG has at least $k - 1$ unserved e_1 -requests at time τ . Moreover, by same reasoning as before, the assumption in Case 2 yields that ALG must have made an empty move if it ends up in node 3 at time τ' , resulting in one extra request for e_1 at time τ' and in $\tau = \tau'$. Otherwise, $p^{\text{ALG}}(\tau') = 1$ and ALG must make an empty move to reach one of e_1 's end points at time $\tau > \tau'$. In either case, ALG has at least k unserved requests for e_1 at time τ .

The adversary serves her pending requests as follows: first, she handles the two pending e_2 -requests, ending up in node 1 at time T_{k+2} . Now, as this is the main difference to Case 1, she must also make an empty move before she can start to serve the e_1 -requests at time T_{k+3} . This empty move can be either to node 2 or node 3 and determines whether $p^{\text{ADV}}(\tau) = 2$ or $p^{\text{ADV}}(\tau) = 4$. No matter how the empty move is done, at time τ there will be exactly two pending requests for the adversary at e_1 . The choice is made according to the following rule: Let v denote the position of ALG's server if starting at time τ it would serve all pending e_1 -requests in a row; then the adversary makes the empty move such that $p^{\text{ADV}}(\tau) \neq v$. Thus, at time τ we are in the situation of Case 1 (with a proper shift in indices). $\square$

In order to prove Theorem 1 by applying Claim 1, we construct the beginning of the sequence in such a way that Property P_k is satisfied for some $k \geq 1$.

To this end starting at time 0, the adversary issues one request for edge e_3 in each time unit (directions alternating), until the online server is located either in node 1 or 2 for the first time. Let this time be $t \in \mathbb{N}$. At time 0, the fair adversary can move her server to the source of the first request issued, either 1 or 2, arriving there at time 1 with two pending requests for edge e_3 . Then, she continues to serve one request for e_3 in every time unit until time t .

When ALG reaches 1 or 2 at time t , it must hold that $t \geq 1$, and $t + 1$ unserved requests for e_3 have piled up in the meantime. Thus, Property P_{t+1} holds at time t for some $t \geq 1$.

This completes the proof of the theorem. $\square$

The construction used in the proof of Theorem 1 also works for even stronger restrictions on the adversary. It can be seen that the adversary used above is even non-abusive in the sense of [11]: besides serving requests she only moves to sources of unserved requests. Finally, Theorem 1 also holds for servers of larger, but finite capacity $K > 1$: we simply multiply each request in the sequence by K , that is, we give each request K times.

3 The $F_{\max}$ -OLTsp Against the Fair Adversary: An Easy 2-Competitive Algorithm

We now consider the special case of the $F_{\max}$ -OLDARP where source and destination of each ride coincide: the $F_{\max}$ -OLTSP. The main difference to the previous section is that a server can serve an unlimited number of requests simultaneously if these requests specify all the same node to be visited. It is easy to see that even on the uniform metric space with at least two points the standard unrestricted adversary can construct sequences where it achieves a zero maximum flow time whereas any deterministic online algorithm has a positive flow time for some request, see also [7, 8, 11]. Consequently, there can not be any strictly competitive algorithm. As already mentioned in the introduction allowing an additive constant $b = n$, that is, equal to the number of nodes in the space, allows for a trivial 1-competitive algorithm.

We therefore consider the $F_{\max}$ -OLTSP against the fair adversary.

Theorem 2. *The algorithm first-come first-serve (FCFS) which always serves an oldest unserved request next is strictly 2-competitive against the fair adversary for the $F_{\max}$ -OLTSP on the uniform metric space.*

Before we can prove Theorem 2, we need to establish an elementary lemma.

Lemma 1. *Given a sequence $\sigma = r_1, \dots, r_m$, let σ_i denote the subsequence of σ that contains the first i requests, i.e., $\sigma_i = r_1, \dots, r_i$. Then $\text{OPT}(\sigma_i) \leq \text{OPT}(\sigma)$ for any $i \in \{1, \dots, m\}$, where OPT refers to a fair adversary.*

Proof. Note that, for the standard adversary, the claim is trivial and holds for any subsequence of σ in place of σ_i . For the fair adversary, however, we must be more careful. Removing some request from a sequence can in fact lead to an increased maximum flow time of the fair adversary, as that request might enlarge the space where the adversary is allowed to move to. To see this, assume that the request r that is removed from σ is the first request for node v . When serving the whole sequence σ , the adversary can benefit from r , if another request q for node v is given later on: he can already be waiting in the “allowed” node v when q request is released, thus incurring a smaller flow time for q and thereby possibly also for other requests to follow.

The described construction, however, is the only way how the fair adversary’s maximum flow time can increase by removing a request, and it shows that the removal of a request can only affect the flow times of requests given later. Hence, removing the tail from a sequence cannot increase the fair adversary’s flow time on the preceding requests. $\square$

3.1 Competitiveness of fcfs

This subsection is dedicated to the proof of Theorem 2 on the competitiveness of FCFS. We first give an intuitive description of the proof. It is based on the following two ideas. First, in order to increase FCFS’ flow time while keeping OPT’s maximum flow time stable, the adversary must issue a second request for some node v shortly after the online server has left that node. The offline server in turn must be able to serve some other requests in the meantime and only arrive in v when it is requested the second time. Then, the second request for v requires the online server to move to v once more while the offline server can serve both requests for v simultaneously. The second useful idea is the following: If F^* denotes OPT’s maximum flow time on a given sequence, the optimal offline algorithm must serve request r_i within the time window $[t_i, t_i + F^*]$. Consequently, once FCFS lags behind by F^* time units, meaning that there exists an unserved request for some node w that is older than F^* time units, the optimal offline server must serve that request before the online server, and can therefore not use another request for w to further increase FCFS’ flow time while keeping OPT’s stable.

For the formal proof, we need some further notation. We say that FCFS serves a request r_i for node v *for free*, if it serves an older request r_j for v together with r_i . By $F^{\text{FCFS}}(r_i)$, we denote the flow time of request r_i in FCFS’ schedule. Note that F^* must be greater than 0 for any meaningful request sequence, since the adversary is fair.

Assume the claim is false and FCFS is not 2-competitive. Then there exists a request sequence on which FCFS’ maximum flow time is more than twice as large as the maximum flow time of OPT on that sequence. Among all request sequences with this property, let $\sigma = r_1, \dots, r_m$ be a shortest one with respect to the number of requests. We call this sequence σ a *shortest counterexample*.

Lemma 2. *FCFS does not serve any request in a shortest counterexample σ for free.*

Proof. If there is a request $r \in \sigma$ that FCFS serves for free, then $\tilde{\sigma} := \sigma \setminus \{r\}$ is a shorter sequence on which FCFS incurs the same flow time as on σ . Since r is served for free by FCFS, there must be an older request for the same node. Thus, r does not open up new space for the adversary and cannot be used to incur a smaller flow time on a later request (cf. the proof of Lemma 1). Therefore, the maximum flow time of the fair adversary cannot become larger by removing r from σ .

This contradicts our definition of σ as a shortest sequence on which FCFS has a maximum flow time that is more than twice as large as that of OPT. $\square$

Our choice of $\sigma = r_1, \dots, r_m$ as a shortest sequence on which FCFS incurs a maximum flow time more than twice as large as OPT's maximum flow time gives rise to another observation.

Lemma 3. *Let F^* denote the maximum flow time of OPT on the shortest counterexample σ . Only the last request r_m is served by FCFS with a flow time of more than $2F^*$.*

Proof. If FCFS served a request r_l for some $l < m$ with a flow time of more than $2F^*$, it would achieve a maximum flow time of more than $2F^*$ also on the cut sequence $\sigma_l = r_1, \dots, r_l$. By Lemma 1, the maximum flow time of OPT on σ_l is at most F^* . Again, this contradicts the definition of σ . $\square$

Consider the schedule of FCFS on σ . If FCFS serves some request r_i at time t with flow time $F^{\text{FCFS}}(r_i)$, then it incurs a flow time of at most $F^{\text{FCFS}}(r_i) + 1$ on the request it serves at time $t + 1$, since this request cannot be older than r_i . Moreover, since FCFS serves the last request in the sequence with a flow time of more than $2F^*$, there must be a time after which FCFS serves all requests with flow time at least F^* .

Define T to be the earliest time such that all requests served by FCFS at or after that time are served with a flow time of at least F^* . By the reasoning above, FCFS must be serving some request at each time $t \geq T$, until it serves r_m , since its flow time can increase by at most one from one request to the next one it serves.

Claim 2. *Let r_i be the request in the shortest counterexample σ that FCFS serves at time T . Then, $F^{\text{FCFS}}(r_i) = F^*$, and no request served by FCFS after time T is released before time $T - F^*$.*

Proof. By definition of T , we have that $F^{\text{FCFS}}(r_i) \geq F^*$. Assume that $F^{\text{FCFS}}(r_i) \geq F^* + 1$. Then, the release time of r_i satisfies $t_i \leq T - (F^* + 1)$. Let r_j be the request served by FCFS at time $T - 1$. Since FCFS does not serve any request in σ for free, and as FCFS serves r_j before r_i , it must hold that $t_j \leq t_i$, and we deduce that

$$F^{\text{FCFS}}(r_j) = T - 1 - t_j \geq T - 1 - t_i \geq T - 1 - (T - (F^* + 1)) = F^*,$$

which contradicts our choice of T . Consequently, $F^{\text{FCFS}}(r_i) = F^*$. This in turn implies that $t_i = T - F^*$. If r_k is a request that FCFS serves after time T , that is, after it has served request r_i , then r_k cannot have been released earlier than r_i by construction of FCFS. Hence, $t_k \geq T - F^*$, as claimed. $\square$

Let L be the time at which OPT finishes serving the shortest counterexample σ . For the final proof of Theorem 2, we represent the requests in σ by an $L \times L$ -matrix M as follows:

$$M_{ij} := \begin{cases} 1, & \text{if some request released at time } i \text{ is served at time } j \text{ by OPT,} \\ 0, & \text{otherwise.} \end{cases}$$

Figure 3 displays the structure of the matrix M . Since FCFS serves no request in σ for free, the same node cannot have been requested twice at the same time. Hence, each request is represented by exactly one non-zero entry of the matrix, and the non-zero entries of column $(\cdot, j)$ of M stand for requests specifying the same node (otherwise, OPT could not serve them simultaneously at time j). Moreover, since OPT's maximum flow time on σ equals F^* , we know that $M_{ij} = 1$ only if $i \leq j \leq i + F^*$. We use M to prove the following claim, which

Fig. 3. The structure of the matrix M : entries $*$ $\in \{0, 1\}$, all other entries are zero.

is the key ingredient for the proof of Theorem 2.

Claim 3. FCFS finishes serving the shortest counterexample σ no later than at time $L + F^*$.

Proof. Let R denote the set of requests served by FCFS at or after time T . By Claim 2, the requests in R are all released at or after time $T - F^*$ and thus served by OPT not earlier than time $T - F^*$. Therefore, by construction of the matrix M , the non-zero entries representing the requests in R are all contained in columns $(\cdot, T - F^*), \dots, (\cdot, L)$. As mentioned above, all requests belonging to the same column specify the same node. Hence, if we are able to show that FCFS serves all requests in R that are contained in the same column simultaneously, then the claim follows: there are $L - (T - F^*) + 1$ columns, and FCFS starts serving the requests in R at time T . Hence, it will be finished at time $L + F^*$ if it serves one column at a time.

To see that FCFS serves all requests in R represented in the same column simultaneously, recall that all requests corresponding to a non-zero entry of column $(\cdot, j)$ must have been released between time $j - F^*$ and j .

By definition of T , the requests in R are served with a flow time of at least F^* . In particular, that request from R in column $(\cdot, j)$ which is served first by FCFS can be served earliest at time $j - F^* + F^* = j$, since its release time is at least $j - F^*$. But at time j , all requests in column $(\cdot, j)$ have already been released, and can therefore be served simultaneously by FCFS. Thus, FCFS indeed serves all requests in R belonging to the same column simultaneously. $\square$

We are now ready to derive the necessary contradiction that proves Theorem 2. Our assumption was that FCFS has a maximum flow time of at least $2F^* + 1$ on σ . By Claim 3, we know that FCFS incurs its maximum flow time on the last request in the sequence, r_m . Since OPT finishes serving at time L , the last request r_m cannot have been released before time $L - F^*$. Otherwise, OPT would incur a flow time of more than F^* on the request it serves at time L , since that request is at least as old as r_m . Thus, by Claim 3, the flow time that FCFS incurs on r_m is at most $L + F^* - (L - F^*) = 2F^*$, contradicting our initial assumption. Consequently, FCFS is 2-competitive. This proves Theorem 2. $\square$

The above results show that the fairness condition restricts the adversary's power sufficiently if the server only needs to visit points. Moreover, note that fairness is not required anymore if we do not ask for strict competitiveness, i.e., if we allow an additive constant $b \geq 1$ in the definition of competitiveness. In fact, the proof remains valid in this case, as can be easily checked.

3.2 A general lower bound

Theorem 3. *For the $F_{\max}$ -OLTSP on a uniform metric space with at least five nodes, no deterministic online algorithm can be strictly c -competitive against the fair adversary with $c < 2$.*

Proof. Let ALG be an arbitrary deterministic online algorithm. The origin is assumed to be in node v_0 . Consider the following instance. First, the adversary gives a request for v_1 at time 0, and a request for v_2 at time 1. Clearly, at time 3, ALG has distance at least 1 to at least one of the nodes in $\{v_0, v_1, v_2\}$. Let $y \in \{v_0, v_1, v_2\}$ be that node. Similarly, there exists a node $z \in \{v_3, v_4, v_5\}$

such that $d(p^{\text{ALG}}(3), z) \geq 1$. Then, at time 3, the adversary issues two more requests: one for y and one for z . Thus, we have that $\sigma = r_1, r_2, r_3, r_4 := (0, v_1), (1, v_2), (3, y), (3, z)$. Figure 4 shows the requests of σ as points of a time-space diagram for $y = v_1$ and $z = v_3$.

Fig. 4. The lower bound construction for the $F_{\max}$ -OLTSP against a fair adversary.

By construction, ALG has distance at least 1 to both nodes y and z at time 3. Therefore, it can have finished serving the last of those two requests earliest at time 5. Since both y and z have been released at time 3, ALG's maximum flow time is at least 2. On the other hand, the adversary can serve request r_1 at time 1, request r_2 at time 2, and then move immediately to y , serving r_3 at time 3 and r_4 at time 4. This gives $\text{OPT}(\sigma) = 1$, which means that ALG is not better than 2-competitive. Note that the adversary is indeed fair: at time 2, the “allowed” subgraph is induced by v_0 (the origin), v_1 and v_2 . Hence, OPT may move to y immediately after having served r_2 . $\square$

References

1. N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer, 2000.
2. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line traveling salesman. *Algorithmica*, 29(4):560–581, 2001.
3. M. Blom, S. O. Krumke, W. E. de Paepe, and L. Stougie. The online-TSP against fair adversaries. *Inform. Journal on Computing*, 13(2):138–148, 2001. A preliminary version appeared in the Proceedings of the 4th Italian Conference on Algorithms and Complexity, 2000, vol. 1767 of *Lecture Notes in Computer Science*.
4. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

5. W. E. de Paepe, J. K. Lenstra, J. Sgall, R. A. Sitters, and L. Stougie. Computer-aided complexity classification of dial-a-ride problems. *Inform Journal on Computing*, 2003. To appear.
6. E. Feuerstein and L. Stougie. On-line single server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.
7. M. Grötschel, S. O. Krumke, and J. Rambau, editors. *Online Optimization of Large Scale Systems*. Springer, Berlin Heidelberg New York, 2001.
8. D. Hauptmeier, S. O. Krumke, and J. Rambau. The online dial-a-ride problem under reasonable load. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2000.
9. S. Irani, X. Lu, and A. Regan. On-line algorithms for the dynamic traveling repair problem. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 517–524, 2002.
10. S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. News from the on-line traveling repairman. *Theoretical Computer Science*, 295(1–3):279–294, 2003. A preliminary version appeared in the Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science, 2001, vol. 2136 of *Lecture Notes in Computer Science*.
11. S. O. Krumke, L. Laura, M. Lipmann, A. Marchetti-Spaccamela, W. E. de Paepe, D. Poensgen, and L. Stougie. Non-abusiveness helps: An $O(1)$ -competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2002.