DIETRICH HAUPTMEIER SVEN O. KRUMKE JÖRG RAMBAU

The Online Dial-a-Ride Problem under Reasonable Load

ZIB-Report 99-08 (November 1999, Revised Version from March 2000)

THE ONLINE DIAL-A-RIDE PROBLEM UNDER REASONABLE LOAD

DIETRICH HAUPTMEIER[1], SVEN O. KRUMKE[1], AND JÖRG RAMBAU[1]

ABSTRACT. In this paper, we analyze algorithms for the online dial-a-ride problem with request sets that fulfill a certain worst-case restriction: roughly speaking, a set of requests for the online dial-a-ride problem is reasonable if the requests that come up in a sufficiently large time period can be served in a time period of at most the same length. This new notion is a stability criterion implying that the system is not overloaded.

The new concept is used to analyze the online dial-a-ride problem for the minimization of the maximal resp. average flow time. Under reasonable load it is possible to distinguish the performance of two particular algorithms for this problem, which seems to be impossible by means of classical competitive analysis.

1. INTRODUCTION

It is a standard assumption in mathematics, computer science, and operations research that problem data are given. However, many aspects of life are online. Decisions have to be made without knowing future events relevant for the current choice. Online problems, such as vehicle routing and control, management of call centers, paging and caching in computer systems, foreign exchange and stock trading, had been around for a long time, but no theoretical framework existed for the analysis of online problems and algorithms.

Meanwhile, competitive analysis has become the standard tool to analyze online-algorithms [4, 6]. Often the online algorithm is supposed to serve the requests one at a time, where a next request becomes known when the current request has been served. However, in cases where the requests arrive at certain points in time this model is not sufficient. In [3, 5] each request in the request sequence has a release time. The sequence is assumed to be in non-decreasing order of release times. This model is sometimes referred to as the *real time model*. A similar approach was used in [1] to investigate the *online dial-a-ride problem*—OLDARP for short— which is the example for the new concept in this paper.

Since in the real time model the release of a new request is triggered by a point in time rather than a decision of the online algorithm we essentially do not need a total order on the set of requests. Therefore, for the sake of convenience, we will speak of *request sets* rather than request sequences.

In the problem OLDARP objects are to be transported between points in a given metric space X with the property that for every pair of points $(x,y) \in X$ there is a path $p : [0,1] \to X$ in X with $p(0) = x$ and $p(1) = y$ of length $d(x,y)$. An important special case occurs when X is induced by a weighted graph.

[1]Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization, Takustr. 7, D-14195 Berlin-Dahlem, Germany. Email: {hauptmeier,krumke,rambau}@zib.de

Research supported by the German Science Foundation (grant 883/5-3).

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 starts and ends its work at a designated origin vertex and which moves along the paths in G. The server picks up and drops objects at their starts and destinations. It is assumed that neither the release time of any future request nor the number of requests is known in advance.

A feasible solution to an instance of the OLDARP is a schedule of moves (i.e., a sequence of consecutive moves in X together with their starting times) in X so that every request is served and that no request is picked up before its release time. The goal of OLDARP is to find a feasible solution with "minimal cost", where the notion of "cost" depends on the objective function used.

Recall that an online-algorithm A is called *c-competitive* if there exists a constant c such that for any request set σ (or request sequence σ if we are concerned with the classical online model) the inequality $\mathrm{A}(\sigma) \leq c \cdot \mathrm{OPT}(\sigma)$ holds. Here, $\mathrm{X}(\sigma)$ denotes the objective function value of the solution produced by algorithm X on input σ and OPT denotes an optimal offline algorithm. Sometimes we are dealing with various objectives at the same time. We then indicate the objective *obj* in the superscript, as in $\mathrm{X}^{obj}(\sigma)$.

Competitive analysis of OLDARP provided the following (see [1]):

- There are competitive algorithms (IGNORE and REPLAN, definitions see below) for the goal of minimizing the *total completion time* of the schedule.
- For the task of minimizing the *maximal (average) waiting time* or the *maximal (average) flow time* there can be no algorithm with constant competitive ratio. In particular, the algorithms IGNORE and REPLAN have an unbounded competitive ratio.

We do not claim originality for the actual online-algorithms IGNORE and REPLAN; instead we show a new method for their analysis. As the reader will see in the definitions, both REPLAN and IGNORE are straight-forward online strategies based on the ability to solve the offline version of the problem to optimality or a constant-factor approximation thereof.

The first—to the best of our knowledge—occurrence of the strategy IGNORE can be found in the paper by Shmoys, Wein, and Williamson [13]: They show a fairly general result about obtaining competitive algorithms for minimizing the total completion time in machine scheduling problems when the jobs arrive over time: If there is a ρ-approximation algorithm for the offline version, then this implies the existence of a 2ρ-competitive algorithm for the online-version, which is essentially the IGNORE strategy. Recall that a ρ-approximation algorithm is a polynomial algorithm that always finds a solution that is at most ρ times the optimum objective value. The results from [13] show that IGNORE-type strategies are 2-competitive for a number of online-scheduling problems. The strategy REPLAN is probably folklore; it can be found also under different names like REOPT or OPTIMAL.

It should be noted that the corresponding offline-problems with release times (where all requests are known at the start of the algorithm) are NP-hard to solve for the objective functions of minimizing the average or maximal flow time—it is even NP-hard to find a solution within a constant factor from the optimum [11]. The offline problem without release times of minimizing the total completion time is polynomially solvable on special graph classes but NP-hard in general [8, 2, 7, 10].

If we are considering a continuously operating system with continuously arriving requests (i.e., the request set may be infinite) then the total completion time is meaningless. Bottom-line: in this case, the existing positive results cannot be applied and the negative results tell us that we cannot hope for performance guarantees that may be relevant in practice. In particular, the two algorithms IGNORE and REPLAN cannot be distinguished by classical competitive analysis.

The point here is that we do not know any notion from the literature to describe what a particular set of requests should look like in order to allow for a continuously operating system. In queuing theory this is usually modelled by a stability assumption: the rate of incoming requests is at most the rate of requests served. To the best of our knowledge, so far there has been nothing similar in the existing theory of discrete online-algorithms. Since in many instances we have no exploitable information about the distributions of requests we want to develop a worst-case model rather than a stochastic model for stability of a continuously operating system.

Our idea is to introduce the notion of Δ-*reasonable* request sets. A set of requests is Δ-reasonable if—roughly speaking—requests released during a period of time $\delta \geq \Delta$ can be served in time at most δ. A set of requests R is *reasonable* if there exists a $\Delta < \infty$ such that R is Δ-reasonable. That means, for non-reasonable request sets we find arbitrarily large periods of time where requests are released faster than they can be served—even if the server has the optimal offline schedule. When a system has only to cope with reasonable request sets, we call this situation *reasonable load*. Section 3 is devoted to the exact mathematical setting of this idea.

We now present our main result on the OLDARP under Δ-reasonable which we prove in Sections 4 and 5.

Theorem 1.1. *For the* OLDARP *under* Δ*-reasonable load,* IGNORE *yields a maximal and an average flow time of at most* 2Δ*, whereas the maximal and the average flow time of* REPLAN *are unbounded.*

The algorithms IGNORE and REPLAN have to solve a number of offline instances of OLDARP, which is in general NP-hard, as we already remarked. We will show how we can derive results for IGNORE when using an approximate algorithm for solving offline instances of OLDARP (for approximation algorithms for offline instances of OLDARP, refer to [8, 2, 7, 10]). For this we refine the notion of reasonable request sets again, introducing a second parameter that tells us, how "fault tolerant" the request set is. In other words, the second parameter tells us, how "good" the algorithm has to be to show stable behavior. Again, roughly speaking, a set of requests is (Δ,ρ)-reasonable if requests released during a period of time $\delta \geq \Delta$ can be served in time at most δ/ρ. If $\rho = 1$, we get the notion of Δ-reasonable as described above. For $\rho > 1$, the algorithm is allowed to work "sloppily" (e.g., employ approximation algorithms) or have break-downs to an extent measured by ρ and still show a stable behavior.

2. PRELIMINARIES

Let us first sketch the problem under consideration. We are given a metric space (X,d). Moreover, there is a special vertex $o \in X$ (the origin). Requests are triples $r = (t,a,b)$, where a is the start point of a transportation task, b its end point, and t its release time, which is—in this context—the time where r becomes known. A *transportation move* is a quadruple $m = (t, a, b, R)$, where a is the starting point and b the end point, and t the starting time, while R is the set (possibly empty) of requests carried by the move. The *arrival time* of a move is the sum of its starting time and $d(a, b)$. A *(closed) transportation schedule* is a sequence $(m_1, m_2, \dots)$ of transportation moves such that

- the first move starts in the origin o;
- the starting point of m_i is the end point of m_{i-1};
- the starting time of m_i carrying R is no earlier than the maximum of the arrival time of m_{i-1} and the release times of all requests in R.
- the last move ends in the origin o.

An *online-algorithm* for OLDARP has to move a server in X so as to fulfill all released transportation tasks without preemption (i.e., once an object has been picked up it is not allowed to be dropped at any other place than its destination), while it does not know about requests that come up in the future. In order to plan the work of the server, the online-algorithm may maintain a preliminary (closed) transportation schedule for all known requests, according to which it moves the server. A posteriori, the moves of the server induce a complete transportation schedule that may be compared to an offline transportation schedule that is optimal with respect to some objective function (competitive analysis). For a detailed set-up see [1].

We start with some useful notation.

Definition 2.1. The *offline version* of $r = (t, a, b)$ is the request

$$r^{\mathit{offline}} := (0, a, b).$$

The *offline version* of R is the request set

$$R^{\mathit{offline}} := \left\{ r^{\mathit{offline}} \, : \, r \in R \right\}.$$

An important characteristic of a request set with respect to system load considerations is the time period in which it is released.

Definition 2.2. Let R be a finite request set for OLDARP. The *release span* $\delta(R)$ of R is defined as

$$\delta(R) := \max_{r \in R} t(r) - \min_{r \in R} t(r).$$

Provably good offline-algorithms exist for the total completion time and the weighted sum of completion times. How can we make use of these algorithms in order to get performance guarantees for minimizing the maximum (average) waiting (flow) times? We suggest a way of characterizing request sets which we want to consider “reasonable”.

3. Reasonable Load

In a continuously operating system we wish to guarantee that work can be accomplished at least as fast as it is presented. In the following we propose a mathematical set-up which models this idea in a worst-case fashion. Since we are always working on finite subsets of the whole request set, the request set itself may be infinite, modeling a continuously operating system.

We start by relating the release spans of finite subsets of a request set to the time we need to fulfill the requests.

Definition 3.1. Let R be a request set for the OLDARP. A weakly monotone function

$$f\colon \begin{cases} \mathbb{R} & \to & \mathbb{R}, \\ \delta & \mapsto & f(\delta); \end{cases}$$

is a *load bound* on R if for any $\delta \in \mathbb{R}$ and any finite subset S of R with $\delta(S) \leq \delta$ the completion time $\textsf{OPT}^{comp}(S^{offline})$ of the optimum schedule for the offline version $S^{offline}$ of S is at most $f(\delta)$. In formula:

$$\textsf{OPT}^{comp}(S^{offline}) \leq f(\delta).$$

Remark 3.2. If the whole request set R is finite then there is always the trivial load bound given by the total completion time of R. For every load bound f we may set $f(0)$ to be the maximum completion time we need for a single request, and nothing better can be achieved.

A "stable" situation would easily obtained by a load bound equal to the identity $x \mapsto x$ on $\mathbb{R}$. (By "stable" we mean that the number of unserved requests in the system does not become arbitrarily large.) In that case we would never get more work to do than we can accomplish. If it has a load bound equal to a function id/ρ, where id is the identity and where $\rho \geq 0$, then ρ measures the tolerance of the request set: assume we have an offline-algorithm at our disposal that is by a factor ρ worse than the optimal offline algorithm then we can still accomplish all the incoming work by using the IGNORE-heuristic: compute a ρ-approximate schedule for the set R of all released but unserved requests. The load bound and the performance guarantee ensure that the schedule takes no longer than $\rho \cdot \Delta(R)/\rho = \Delta(R)$. Thus, the set of requests that are released in the meantime has a release span no larger than $\Delta(R)$, and we can proceed by computing a ρ-approximate schedule for that set.

However, we cannot expect that the identity (or any linear function) is a load bound for OLDARP because of the following observation: a request set consisting of one single request has a release span of 0 whereas in general it takes non-zero time to serve this request. In the following definition we introduce a parameter describing how far a request set is from being load-bounded by the identity.

Definition 3.3. A load bound f is *(Δ,ρ)-reasonable* for some $\Delta, \rho \in \mathbb{R}$ if

$$\rho f(\delta) \leq \delta \quad \text{for all } \delta \geq \Delta$$

A request set R is (Δ,ρ)*-reasonable* if it has a (Δ,ρ)-reasonable load bound. For $\rho = 1$, we say that the request set is Δ-reasonable.

In other words, a load bound is *(Δ,ρ)-reasonable*, if it is bounded from above by $id(x)/\rho$ for all $x \geq \Delta$ and by the constant function with value Δ/ρ otherwise.

Remark 3.4. If Δ is sufficiently small so that all request sets consisting of two or more requests have a release span larger than Δ then the first-come-first-serve strategy is good enough to ensure that there are never more than two unserved requests in the system. Hence, the request set does not require scheduling the requests in order to provide for a stable system.

In a sense, Δ is a measure for the combinatorial difficulty of the request set R. Thus, it is natural to ask for performance guarantees for algorithms in terms of this parameter. This is done for the algorithm IGNORE in the next section.

4. Bounds for the Flow Times of IGNORE

We are now in a position to prove bounds for the maximal resp. average flow time in the OLDARP for algorithm IGNORE stated in Theorem 1.1. We start by recalling the algorithm IGNORE from [1]

Definition 4.1 (Algorithm IGNORE)**.** Algorithm IGNORE works with an internal buffer. It may assume the following states (initially it is IDLE):

IDLE: Wait for the next point in time when requests become available. Goto PLAN.

BUSY: While the current schedule is in work store the upcoming requests in a buffer ("ignore them"). Goto IDLE if the buffer is empty else goto PLAN.

PLAN: Produce a preliminary transportation schedule for all currently available requests R (taken from the buffer) minimizing *comp* for R^{offline}. (Note: This yields a feasible transportation schedule for R because all requests in R are immediately available.) Goto BUSY.

We assume that IGNORE solves offline instances of OLDARP employing a ρ-approximation algorithm.

Let us consider the intervals in which IGNORE organizes its work in more detail. The algorithm IGNORE induces a dissection of the time axis $\mathbb{R}$ in the following way: We can assume, w.l.o.g., that the first set of requests arrives at time 0. Let $\delta_0 = 0$, i.e., the point in time where the first set of requests is released (these are processed by IGNORE in its first schedule). For $i > 0$ let δ_i be the duration of the time period the server is working on the requests that have been ignored during the last δ_{i-1} time units. Then the time axis is split into the intervals

$$[\delta_0 = 0, \delta_0], (\delta_0, \delta_1], (\delta_1, \delta_1 + \delta_2], (\delta_1 + \delta_2, \delta_1 + \delta_2 + \delta_3], \dots$$

Let us denote these intervals by $I_0, I_1, I_2, I_3, \dots$. Moreover, let R_i be the set of those requests that come up in I_i. Clearly, the complete set of requests R is the disjoint union of all the R_i.

At the end of each interval I_i we solve an offline problem: all requests to be scheduled are already available. The work on the computed schedule starts immediately (at the end of interval I_i) and is done δ_{i+1} time units later (at the end of interval I_{i+1}). On the other hand, the time we need to serve the schedule is not more than ρ times the optimal completion time of ${R_i}^{\mathit{offline}}$. In other words:

Lemma 4.2. *For all $i \geq 0$ we have*

$$\delta_{i+1} \leq \rho \cdot \text{OPT}^{comp}({R_i}^{\mathit{offline}}).$$

Let us now recall and prove the first statement of Theorem 1.1.

Theorem 4.3. *Let $\Delta > 0$ and $\rho \geq 1$. For all instances of* OLDARP *with (Δ, ρ)-reasonable request sets,* IGNORE *employing a ρ-approximate algorithm for solving offline instances of* OLDARP *yields a maximal flow time of no more than 2Δ.*

Proof. Let r be an arbitrary request in R_i for some $i \geq 0$, i.e., r is released in I_i. By construction, the schedule containing r is finished at the end of interval I_{i+1}, i.e., at most $\delta_i + \delta_{i+1}$ time units later than r was released. Thus, for all $i > 0$ we get that

$$\text{IGNORE}^{maxflow}(R_i) \leq \delta_i + \delta_{i+1}.$$

If we can show that $\delta_i \leq \Delta$ for all $i > 0$ then we are done. To this end, let $f : \mathbb{R} \to \mathbb{R}$ be a (Δ, ρ)-reasonable load bound for R. Then $\textsf{OPT}^{comp}(R_i{}^{\mathit{offline}}) \leq f(\delta_i)$ because $\delta(R_i) \leq \delta_i$.

By Lemma 4.2, we get for all $i > 0$

$$\delta_{i+1} \leq \rho\textsf{OPT}^{comp}(R_i{}^{\mathit{offline}}) \leq \rho f(\delta_i) \leq \max\{\delta_i, \Delta\}.$$

Using $\delta_0 = 0$ the claim now follows by induction on i. □

The statement of Theorem 1.1 concerning the average flow time of IGNORE follows from the fact that the average is never larger than the maximum.

Corollary 4.4. *Let $\Delta > 0$. For all Δ-reasonable request sets algorithm* IGNORE *yields a average flow time no more than 2Δ.*

5. A DISASTROUS EXAMPLE FOR REPLAN

We first recall the strategy of algorithm REPLAN for the OLDARP. Whenever a new request becomes available, REPLAN computes a preliminary transportation schedule for the set R of all available requests by solving the problem of minimizing the total completion time of R^{offline}.

Then it moves the server according to that schedule until a new request arrives or the schedule is done. In the sequel, we provide an instance of OLDARP and a Δ-reasonable request set R such that the maximal and the average flow time $\textsf{REPLAN}^{\mathit{maxflow}}(R)$ is unbounded, thereby proving the remaining assertions of Theorem 1.1.

Theorem 5.1. *There is an instance of* OLDARP *under reasonable load such that the maximal and the average flow time of* REPLAN *is unbounded.*

Proof. In Figure 1 there is a sketch of an instance for the OLDARP. The metric space is a path on four nodes a, b, c, d with origin a; the length of the path is ℓ, the distances are $d(a,b) = d(c,d) = \varepsilon$, and hence $d(b,c) = \ell - 2\varepsilon$. At time 0 a request from a to d is issued; at time $3/2\ell - \varepsilon$, the remaining requests periodically come in pairs from b to a and from c to d, resp. The time distance between them is $\ell - 2\varepsilon$.

We show that for $\ell = 18\varepsilon$ the request set R indicated in the picture is $2\frac{2}{3}\ell$-reasonable. Indeed: it is easy to see that the first request from a to d does not influence reasonability. Consider an arbitrary set R_k of k adjacent pairs of requests from b to a resp. from c to d. Then the release span $\delta(R_k)$ of R_k is

$$\delta(R_k) = (k-1)(\ell - 2\varepsilon).$$

The offline version $R_k{}^{\mathit{offline}}$ of R_k can be served as follows: first, move the server to c, the starting point of the upper requests: this contributes cost $\ell - \varepsilon$. Next, serve all the upper requests and go back to c: this contributes cost $k \times 2\varepsilon$. Then, go down to b, the starting point of the lower requests: this contributes another $\ell - 2\varepsilon$ to the cost. Now, serve the first lower requests: the additional cost for this is ε. Finally, serve the remaining lower requests at an additional cost of $(k-1) \cdot 2\varepsilon$. In total, we have the following:

$$\textsf{OPT}^{comp}(R_k{}^{\mathit{offline}}) = 2\ell + (k-1) \cdot 4\varepsilon.$$

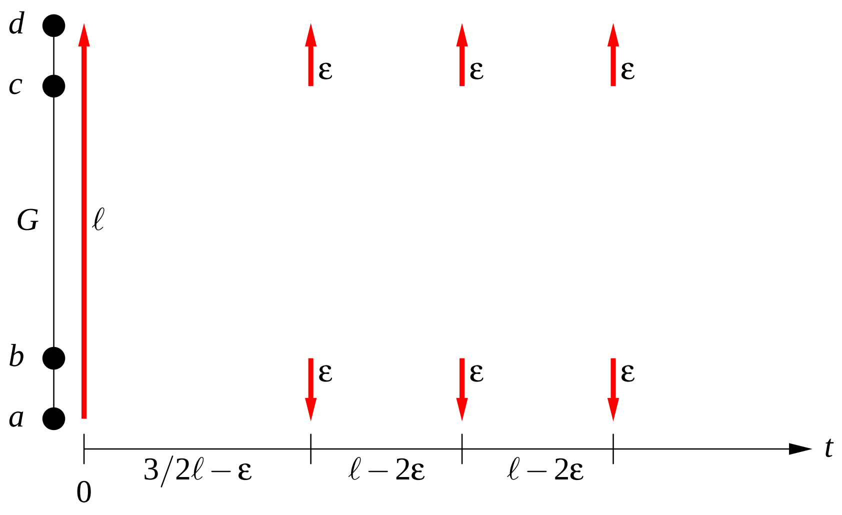

FIGURE 1. A sketch of a $(2\frac{2}{3}\ell)$-reasonable instance of OLDARP $(\ell = 18\varepsilon)$. The horizontal axis holds the time, the vertical axis depicts the metric space in which the server moves. A request is denoted by an arrow from its starting point to its end point horizontally positioned at its release time.

In order to find the smallest paramter Δ for which the request set R_k is Δ-reasonable we solve for the integer $k-1$ and get

$$k-1 = \left\lceil \frac{2\ell}{\ell - 6\varepsilon} \right\rceil = 3.$$

Hence, we can set Δ to

$$\Delta := \text{OPT}^{comp}(R_4{}^{offline}) = 2\tfrac{2}{3}\ell.$$

Now we define

$$f : \begin{cases} \mathbb{R} & \rightarrow & \mathbb{R}, \\ \delta & \mapsto & \begin{cases} \Delta & \text{for } \delta < \Delta, \\ \delta & \text{otherwise.} \end{cases} \end{cases}$$

By construction, f is a load bound for R_4. Because the time gap after which a new pair of requests occurs is certainly larger than the additional time we need to serve it (offline), f is also a load bound for R. Thus, R is Δ-reasonable, as desired.

Now: how does REPLAN perform on this instance? In Figure 2 we see the track of the server following the preliminary schedules produced by REPLAN on the request set R.

The maximal flow time of REPLAN on this instance is realized by the flow time of the request $(3/2\ell - \varepsilon, b, a)$, which is unbounded.

Moreover, since all requests from b to a are postponed after serving all the requests from c to d we get that REPLAN produces an unbounded average flow time as well. □

In Figure 3 we show the track of the server under the control of the IGNORE-algorithm. After an initial inefficient phase the server ends up in a stable operating mode. This example also shows that the analysis of IGNORE in Section 4 is sharp.

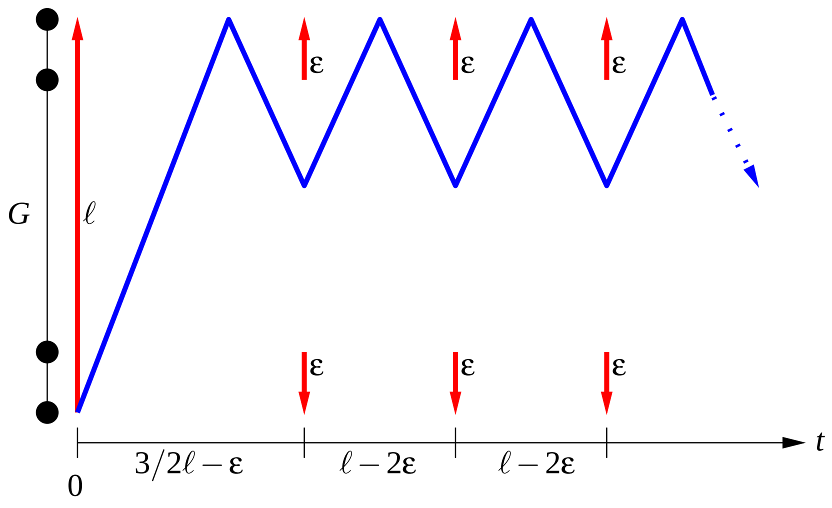

FIGURE 2. The track of the REPLAN-server is drawn as a line in the diagram: at each point in time t we can read off the position of the server by looking at the height of the line at the horizontal position t. Because a new pair of requests is issued exactly when the server is still closer to the requests at the top all the requests at the bottom will be postponed in an optimal preliminary schedule. Thus, the server always returns to the top when a new pair of requests arrives.

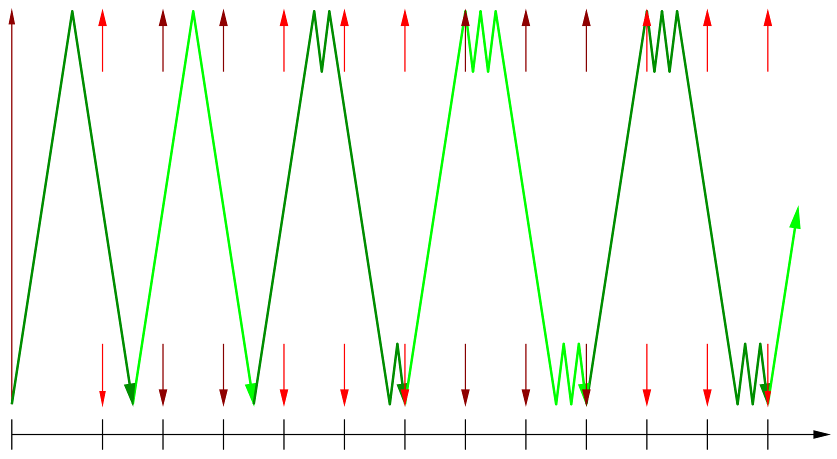

FIGURE 3. The track of the IGNORE-server.

6. HOW ABOUT REPLANNING WITH A DIFFERENT OBJECTIVE?

It is quite a natural question to ask whether modified replan strategies FLOWREPLAN or MAXFLOWREPLAN that repeatedly minimizes the average resp. maximal flow times on the known request sets would give a reasonable bound on the maximal and average flow times in the online situation.

We mentioned already that the offline problem of minimizing the average flow time is very hard. In the offline problem that FLOWREPLAN has to solve, however,

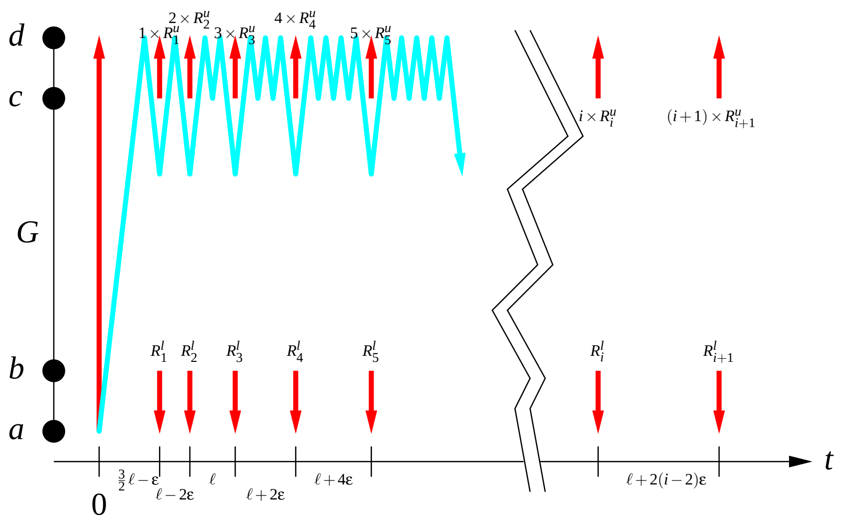

FIGURE 4. The track of the FLOWREPLAN-server on a the example from Theorem 6.1.

all requests have release times in the past. It is then easy to see that the problem is equivalent to the minimization of the average completion time counted from the point in time where the planning takes place. Moreover, since the average flow time is larger by the "average age" of the requests, the performance guarantees of approximation algorithms minimizing the average completion time carry over. Still, in our computational experience minimization of the average completion time takes more time than minimizing the total completion time.

Anyway: the following result shows that even under reasonable load we cannot expect a worst case stable behaviour of FLOWREPLAN.

Theorem 6.1. *There is an instance of* OLDARP *under reasonable load such that the maximal and average flow times of* FLOWREPLAN *are unbounded.*

Proof. We construct a set of requests in the same metric space as in the previous Section 5 as follows:

- At time 0 we issue again one request from a to d.
- At time $T_0 := 3/2\ell - \varepsilon$ we issue a pair of requests R_1^u from c to d and R_1^l from b to a.
- At time $T_{i+1} := T_i + \ell + 2(i-2)\varepsilon$ we issue
  - a set of i "upper" requests R_{i+1}^u from c to d and
  - one "lower" request R_{i+1}^l from b to a.

Figure 4 sketches the construction.

For $\ell = 18\varepsilon$ this request set is again $2\frac{2}{3}\ell$-reasonable since we have increased the time intervals between the release times of the requests by the additional amount that is needed to serve the additional copies of upper requests.

At time T_i, for all $i > 0$, FLOWREPLAN has still to serve as many upper requests as there are lower requests. Thus, at T_i the schedule with minimum average flow time for the currently available requests serves the upper requests first. Hence, the requests at the bottom have to wait for an arbitrarily long period of time.

In order to prove the assertion concerning the average flow time we consider the result $f(R_N)$ that FLOWREPLAN produces on the input set R_N which contains all requests up to time T_N.

The sum of all flow times $f_\Sigma(R_N)$ is dominated by the waiting times of the lower requests. That is, it is a least

$$\begin{aligned} f_\Sigma(R_N) &\geq \sum_{k=1}^{N}\sum_{i=k}^{N}(\ell+2(i-2)\varepsilon) \\ &\geq \sum_{k=1}^{N}\sum_{i=k}^{N}(i-2)\varepsilon. \end{aligned}$$

The number of requests $\#R_N$ in R_N is

$$\#R_N = 1+\sum_{k=1}^{N}(k+1),$$

so that

$$f(R_N) = \frac{f_\Sigma(R_N)}{\#R_N} \xrightarrow{N\to\infty} \infty,$$

which completes the proof. □

A strategy that minimizes just the maximal flow time does not make a lot of sense since sometimes this only determines which request is to be served first; the order in which all the other requests are scheduled is unspecified. Thus, the most sensible strategy in this respect seems to be the following: consider an offline instance of the dial-a-ride problem. The vector consisting of all flow times of requests in a feasible solution ordered decreasingly is the *flow vector*. All flow vectors are ordered lexicographically. The online strategy MAXFLOWREPLAN for the online dial-a-ride problem does the following: whenever a new request becomes available MAXFLOWREPLAN computes a new schedule of all yet unserved requests minimizing the flow vector.

It is an open problem what the performance of this strategy is under Δ-reasonable load. In practice, however, it is probably too difficult to solve the offline problem with this objective function.

7. Reasonable load as a general framework

We introduced the new concept of reasonable request sets, using as example the problem OLDARP. However, the concept can be applied to any combinatorial online problem with (possibly infinte) sets of time stamped requests, such as online scheduling, e.g., as described by Sgall [12], or the Online Traveling Salesman Problem, studied by Ausiello et.al. [3].

The algorithms IGNORE and REPLAN represent general “online paradigms” which can be used for any online problem with time-stamped requests. We notice that the proof of the result that the average and maximal flow and waiting times of IGNORE are bounded by 2Δ has not explicetly drawn on any specific property of OLDARP—this result holds for all combinatorial online problems with requests given by their release times.

The proof that the maximal flow and waiting time of a Δ-reasonable request set is unbounded for REPLAN is equally applicable to the Online Traveling Salesman Problem by Ausiello et.al. [3]. We expect that the same is true for any “sufficiently difficult" online problem with release times—for very simple problems, such as OLDARP on a zero dimensional space, the result trivially does not hold.

8. CONCLUSION

We have introduced the mathematical notion Δ-reasonable describing the combinatorial difficulty of a possibly infinite request set for OLDARP. For reasonable request sets we have given bounds on the maximal resp. average flow time of algorithm IGNORE for OLDARP; in contrast to this, there are instances of OLDARP where algorithm REPLAN yields an unbounded maximal and average flow time. One key property of our results is that they can be applied in continuously working systems. Computer simulations have meanwhile supported the theoretical results in the sense that algorithm IGNORE does not delay individual requests for an arbitraryly long period of time, whereas REPLAN has a tendency to do so [9].

While the notion of Δ-reasonable is applicable to minimizing maximal flow time, it would be of interest to investigate an average analogue in order to prove non-trivial bounds for the average flow times.

REFERENCES

[1] Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau, *The online transportation problem*, Preprint SC 98-34, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1998.

[2] Mikhail J. Atallah and S. Rao Kosaraju, *Efficient solutions to some transportation problems with applications to minimizing robot arm travel*, SIAM Journal on Computing **17** (1988), 849–869.

[3] Georgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio 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.

[4] Allan Borodin and Ran El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.

[5] Esteban Feuerstein and Leen Stougie, *On-line single server dial-a-ride problems*, Manuscript, submitted for publication, 1998.

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

[7] Greg N. Frederickson and D. J. Guan, *Nonpreemptive ensemble motion planning on a tree*, Journal of Algorithms **15** (1993), 29–60.

[8] Greg N. Frederickson, Matthew S. Hecht, and Chul Kim, *Approximation algorithms for some routing problems*, SIAM Journal on Computing **7** (1978), 178–193.

[9] Martin Grötschel, Dietrich Hauptmeier, Sven O. Krumke, and Jörg Rambau, *Simulation studies for the online dial-a-ride-problem*, Preprint SC 99-09, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1999.

[10] Dietrich Hauptmeier, Sven O. Krumke, Jörg Rambau, and Hans C. Wirth, *Euler is standing in line—dial-a-ride problems with fifo-precedence-contraints*, Preprint SC 99-06, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1999.

[11] Hans Kellerer, Thomas Tautenhahn, and Gerhard Woeginger, *Approximability and nonapproximabiblity results for minimizing total flow time on a single machine*, Proceedings of the Symposium on the Theory of Computing, 1996.

[12] Jiří Sgall, *Online scheduling*, Online Algorithms: The State of the Art (Amos Fiat and Gerhard J. Woeginger, eds.), Lecture Notes in Computer Science, vol. 1442, Springer, 1998.

[13] David B. Shmoys, Joel Wein, and David P. Williamson, *Scheduling parallel machines on-line*, SIAM Journal on Computing **24** (1995), no. 6, 1313–1331.

{DIETRICH HAUPTMEIER, SVEN O. KRUMKE, JÖRG RAMBAU}, KONRAD-ZUSE-ZENTRUM FÜR INFORMATIONSTECHNIK BERLIN, TAKUSTR. 7, 14195 BERLIN, GERMANY

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

URL: http://www.zib.de/{hauptmeier,krumke,rambau}