

# Serving Requests That Arrive on a Graph as to Minimize Flowtime

Axel Simroth <sup>\*</sup>      Alexander Souza <sup>†</sup>

October 24, 2007

## Abstract

We study an online problem in which a server with certain speed  $s$  operates on a weighted graph and has to serve requests to the vertices that arrive at some rate. Each request takes constant time  $\varepsilon > 0$  to be served and the objective is to minimize the total flowtime. We consider both deterministic (restricted and unrestricted) and randomized adversaries.

It turns out that the competitive ratio of a specific algorithm depends on the speed of the server. Even against an unrestricted adversary, if the server is fast in comparison to the diameter of the graph, then the competitive ratio is  $1 + o(1)$ . For slow servers we give the lower bound  $\Omega\left(\frac{h}{s}\right)$ , where  $h$  is the length of a shortest (not necessarily simple) circuit in the graph that visits all vertices. For a restricted adversary, which is forced to “spread” the requests in the graph, we give a bound on the competitive ratio of the algorithm that decreases as the diffusion grows. The bound ranges from matching the lower bound down to a constant. Furthermore we consider a randomized adversary where  $0 < p \leq 1$  is a probabilistic parameter of request-diffusion. Then the competitive ratio is at most the bound against the deterministically restricted adversary up to a factor of  $O(p^{-1})$ .

Additionally we show that serving a set of known requests can be solved optimally in polynomial time if the underlying graph is a line.

## 1 Introduction

In many optimization problems in operations research and computer science, one has to deal with data inherently becoming available over time. Well-known examples of these so-called *online* phenomena are paging and caching, list update, scheduling with jobs arriving over time, online traveling salesman, and dial-a-ride problems. All these problems have in common that a sequence of requests (items, jobs, locations, commodities) becomes available one-by-one. They differ in the way these requests need to be processed.

In a seminal work, Sleator and Tarjan [9] introduced a – by now classical – approach to study these sort of problems: *competitive analysis*. In this model, the *competitive ratio* is the main measure of interest. This is the largest possible ratio of the objective function value achieved by an online algorithm in comparison with the optimum value of the whole instance.

This classical competitive analysis was often criticized for being overly unfair since an algorithm without foresight is facing an almighty malicious adversary. Not surprisingly, such

---

<sup>\*</sup>Fraunhofer Institut for Transportation and Infrastructure Systems IVI, Dresden, Germany, [axel.simroth@ivi.fraunhofer.de](mailto:axel.simroth@ivi.fraunhofer.de)

<sup>†</sup>Department of Computer Science, University of Freiburg, Germany, [souza@informatik.uni-freiburg.de](mailto:souza@informatik.uni-freiburg.de)

an adversary can frequently force algorithms to bad decisions – even those algorithms that are desirable from a practical perspective. Hence modeling “reasonably” restricted adversaries was – and continues to be – an important topic in online computation; see Koutsoupias and Papadimitriou [6] for a discussion. In this paper we follow this idea and investigate an online problem which is closely related to the work of Hauptmeier et al. [5], where our main interest is in “plausible” restrictions in the structure of the sequences admissible for the adversary.

We consider the problem that a server with certain speed  $s$  operates on a graph with positive edge-weights and a sequence of requests to its vertices becomes available at a certain rate. In order to handle a request, the server has to reach the associated vertex and each serving takes constant time  $\varepsilon > 0$ . The objective is to minimize the total flowtime of the requests. Here *flowtime* denotes the time between the arrival of a request and its serving, i.e., the time the request spends in the system.

It was already observed in [5] that the competitive ratio exists if and only if the serving of at least one request takes strictly positive time. The underlying problem is that the optimum value might be zero otherwise (in which case the competitive ratio is meaningless). Among other things, in [5] bounds on the total (respectively maximum) flowtime of an algorithm called IGNORE were given. However, for the above reason, no competitive analysis was provided.

In our opinion, the assumption that each request takes a certain positive time to be served is rather natural. Hence knowing that the competitive ratio exists, the question of its actual value raises immediately. In this paper, we analyze variants of the IGNORE algorithm and give bounds on their competitive ratios. We study deterministic and randomized adversaries with restrictions concerning the structure of the request sequence. It turns out that the ratio ranges from  $1 + o(1)$  up to linear in the total weight of the graph depending on the restrictions and the speed of the server. We discuss our contribution in further detail below.

## Related Work

Hauptmeier et al. [5] considered an online dial-a-ride problem, where a server operates on a weighted graph and requests arrive over time. Each request requires that the server transports a certain object from a source- to a target-vertex. They restrict the adversary by requiring that the load generated is “reasonable.” This essentially means that requests released during a period  $\delta$  can be served in time at most  $\delta$ . They give bounds on the total (respectively maximum) flowtime of algorithm IGNORE but do not provide competitive analysis.

In the related work of Gutiérrez et al. [4] one can observe the sensitivity of online dial-a-ride problems towards the underlying metric. In their variant, the objective is to serve as many arriving requests as possible before they disappear. If the graph is a line, then no deterministic algorithm can have constant competitive ratio, while on the uniform metric this can be the case.

Apparently, the algorithmic concept behind IGNORE was introduced in work of Shmoys et al. [8] in the context of online makespan minimization. The idea is to schedule the jobs that are available and ignore the jobs arriving in the meantime until the available jobs are completed. Then the previously ignored jobs are scheduled, and so on.

The underlying idea, to wait until several jobs have arrived has also proved useful for online travelling-salesman problems, for example when the underlying graph is a line. The tight bounds proved in the papers [1, 2, 3] state that there is an algorithm which is allowed to wait that has lower competitive ratio than a best-possible non-waiting algorithm.

## Our Results

In our model, a server with certain speed  $s$  operates on a weighted graph and has to serve requests to the vertices that arrive at some (constant) rate. Each request takes constant time  $\varepsilon > 0$  to be served and the objective is to minimize the total flowtime. Generally speaking, we are interested in restricting the admissible sequences for the adversary, such that this choice seems “plausible” and allows to improve competitiveness upon the worst case.

In Section 3 we consider deterministic adversaries, both, restricted and unrestricted. It turns out that the competitive ratio of IGNORE algorithms depends on the speed  $s$  of the server. If  $s$  is large in comparison to the diameter of the graph, then the competitive ratio is  $1 + o(1)$  (Theorem 3.1). Hence no adversarial restriction is needed for these servers.

For slow servers facing an unrestricted adversary we establish the lower bound  $\Omega\left(\frac{h}{s}\right)$ , where  $h$  is the length of a shortest (not necessarily simple) circuit in the graph that visits all vertices (Theorem 3.2). Then we introduce a restriction for the adversary. We essentially require that the requests must not be concentrated on “too few” vertices but have to be spread around a “large portion” of the graph. We model this “diffusion” with two parameters and give a bound on the competitive ratio of IGNORE that decreases as the diffusion grows (Theorem 3.7). The bound ranges from matching the lower bound down to a constant.

In Section 4 we consider a randomized adversary with relaxed restrictions by requiring that subsequences only have to be diffuse with a certain probability  $p$ . It turns out that the competitive ratio against this adversary is at most the bound of the deterministically restricted adversary up to a factor of  $O(p^{-1})$  (Theorem 4.2).

Finally, in Section 5 we give an optimal strategy for the phases of IGNORE for the special case that the underlying graph is a line. We show that for this case an optimal schedule can be computed in polynomial time by solving an appropriate shortest-path problem (Theorem 5.2).

## 2 Preliminaries

Throughout the paper, we consider the following model. A server operates on a connected undirected graph  $G = (V, E)$  with  $n$  vertices  $V = \{1, 2, \dots, n\}$  and non-negative edge-weights  $w(e) \geq 0$ . We define the *distance*  $d_{i,j}$  between two vertices  $i, j \in V$  to be the length of a shortest path connecting  $i$  with  $j$ . We define the *diameter* of  $G$  by  $d = \max_{i,j} d_{i,j}$ . Further, we define the *hamiltonicity* of  $G$  as the length  $h$  of a shortest, not necessarily simple, path that visits all vertices and returns to its origin.

The server moves at a certain *speed*  $s > 0$  which may depend on  $n$ . Hence the time to pass an edge  $e \in E$  is  $\frac{w(e)}{s}$ . At the points in time  $T = \{1, 2, \dots, t\}$  requests  $R = (r_1, r_2, \dots, r_t)$  to vertices arrive, i.e., each  $r_i \in V$ . Once the server has reached the vertex  $r_i$  the request requires constant *service-time*  $0 < \varepsilon < 1$ . We point out that the assumption  $\varepsilon < 1$  is there only to avoid a degenerate situation: if each request takes longer to be served than the next one arrives, this will lead to congestion no matter how fast the server is. This assumption parallels the concept of *reasonable load* in [5].

The point in time when  $r_i$  becomes known is called the *arrival*  $a_i$  of  $r_i$ , the time when it is completely served, its *departure*  $d_i$ . Its *flowtime* is  $f_i = d_i - a_i$ , i.e., the time the request has to wait in the system until served.

A *schedule*  $S$  is a collection of pairs  $S = ((r_1, t_1), (r_2, t_2), \dots, (r_t, t_t))$ , where for each  $i = 1, \dots, t$  the component  $r_i$  is a request and  $t_i$  the point in time when it is served. Note that it is not required that the requests be served in the same ordering as they arrive. The

server visits the requests according to its chosen schedule in increasing order of the  $t_i$ . We additionally assume that the server is *lazy*: it stays at its current position as long as possible. For any schedule  $S$ ,  $F(S) = \sum_{i=1}^t f_i(S)$  denotes its *total flowtime*, where  $f_i(S)$  is the flowtime of request  $r_i$  induced by  $S$ . (If no confusion arises, we omit the argument and simply write  $f_i$  instead of  $f_i(S)$ ). A natural objective in this setting is to minimize the total flowtime which we will consider throughout the paper. Furthermore,  $S^*$  denotes the schedule that minimizes the total flowtime given a request sequence  $R$  in advance.

An algorithm ALG is called *online* if its decision at any point in time  $\tau$  may only depend on the past, i.e., the requests  $r_1, \dots, r_i$  with  $i \leq \tau$ . Otherwise the algorithm is called *offline*. By convention, we write  $\text{ALG}(R) = F(S)$  and  $\text{OPT}(R) = F(S^*)$ , where  $S$  and  $S^*$  are the respective schedules produced by ALG and OPT for the requests  $R$ .

## Adversaries

We study two different sources of the request sequence  $R$ . Firstly, we consider a deterministic adversary  $A_{P,t}$ , which is explicitly restricted in his choice of the request sequence: Let  $P$  be any predicate and let  $t$  be an integer, then the adversary  $A_{P,t}$  is free to choose any request sequence in the set  $R_{P,t} = \{R = (r_1, r_2, \dots, r_t) : R \text{ satisfies } P\}$ . Following the competitive paradigm, we measure the quality of schedules produced by an algorithm ALG facing an adversary  $A_{P,t}$  with the *competitive ratio*

$$C_{P,t}^{\text{ALG}} = \max_{R \in R_{P,t}} \frac{\text{ALG}(R)}{\text{OPT}(R)}.$$

If the adversary is not restricted, then we write  $C_t^{\text{ALG}}$ . Secondly, following the *diffuse adversary* concept of Koutsoupias and Papadimitriou [6], we define a randomized adversary  $\Delta_{P,t}$  as follows. Let  $D$  be any probability distribution on the set of request sequences of length  $t$  and let  $P$  be a predicate. Let the set  $D_{P,t}$  denote all the distributions  $D$  that satisfy the predicate  $P$ . The adversary  $\Delta_{P,t}$  is allowed to choose any distribution in  $D_{P,t}$ . In contrast to [6], we define the expected competitive ratio by

$$E_{P,t}^{\text{ALG}} = \max_{D \in D_{P,t}} \mathbb{E} \left[ \frac{\text{ALG}}{\text{OPT}} \right].$$

Here ALG and OPT denote the respective random variables induced by  $D$  on  $\text{ALG}(R)$  and  $\text{OPT}(R)$ . If no restriction is imposed on  $D$ , then we write  $E_t^{\text{ALG}}$  and observe  $E_t^{\text{ALG}} = C_t^{\text{ALG}}$ , i.e., the probabilistic and the deterministic model can be identical. Our definition differs from the one given in [6]. They defined the expected competitive ratio to be  $\max_{D \in D_{P,t}} \frac{\mathbb{E}[\text{ALG}]}{\mathbb{E}[\text{OPT}]}$ . The reason why we have chosen to consider  $\mathbb{E} \left[ \frac{\text{ALG}}{\text{OPT}} \right]$  is because  $\frac{\text{ALG}}{\text{OPT}}$  is an *instance-wise* comparison of the (random) objective value achieved by an algorithm with the optimum solution. In our opinion, this reflects the spirit of competitive analysis more naturally. In fact the two measures can behave substantially different; see [7] for an example.

## IGNORE Algorithms

Throughout the paper, we consider a family of algorithms, called IGNORE algorithms. An IGNORE algorithm is characterized by maintaining two memories  $M$  and  $M'$  that are served in *phases* as follows. Suppose that the algorithm has collected several requests in the memory  $M$  and has started serving those. Then, all the requests that arrive in the meantime are stored

in the memory  $M'$  (and are hence ignored in the current phase). After  $M$  is served, the phase ends and the procedure repeats with changed roles of  $M$  and  $M'$ . Note that we have not yet specified how the requests in  $M$  are actually served, which leaves space for various strategies.

### 3 Deterministic Adversaries

In this section, we consider deterministic adversaries – restricted and unrestricted. It turns out that it is useful to distinguish between fast and slow servers. We find that fast servers are able to handle any request sequence almost optimally, see Section 3.1 and we show how good slow servers perform depending on the freedom of the adversary, see Section 3.2.

Consider a sequence  $G_1, G_2, \dots$  of graphs, where each  $G_n$  has  $n$  vertices. This induces sequences  $d_1, d_2, \dots$  of diameters and  $h_1, h_2, \dots$  of hamiltonicities. Recall that also the speed of the server may depend on  $n$ , which gives a sequence  $s_1, s_2, \dots$  of speeds.

A server is *fast* if  $s = \omega(d)$ , where the asymptotics is in the number  $n$  of vertices in the above sense. A server is *slow* if  $s = O(h)$ . Since  $d \leq h$  we have that a server may be fast and slow at the same time. Since we show bounds for both cases, the stronger one applies.

#### 3.1 Fast Servers

In this section we show that a trivial IGNORE algorithm is asymptotically optimal if the server is fast enough. Hence no restrictions of the adversary are necessary.

The algorithm ALG works as follows. Initially collect only one single request in the memory  $M$  and serve it. Store the arriving requests in  $M'$  and serve those in the next phase. The motivating observation behind the algorithm is that, for a sufficiently fast server, at most one request will have arrived until the requests of the current phase are served. Hence the algorithm serves the requests in the order in which they arrive. If the server is fast enough, this is already almost optimal, which is stated in the following result.

**Theorem 3.1.** *For a fast server, i.e.,  $s = \omega(d)$ , we have  $C_t^{\text{ALG}} = 1 + o(1)$ .*

*Proof.* Observe that the flowtime of a request  $r_i$  is at most  $f_i \leq \frac{d}{s} + \varepsilon = o(1) + \varepsilon$  given that the server is fast enough so that only one request arrives while the server is busy serving the current phase. Taking the sum over all  $t$  requests yields  $\text{ALG} \leq t\varepsilon + o(t)$ . With the trivial lower bound  $\text{OPT} \geq t\varepsilon$  we have that for every sequence  $R$   $\frac{\text{ALG}(R)}{\text{OPT}(R)} \leq \frac{t\varepsilon + o(t)}{t\varepsilon} = 1 + o(1)$ , where we have used that  $\varepsilon$  is a constant. ■

#### 3.2 Slow Servers

In this section, we consider the remaining case: slow servers, i.e.,  $s = O(h)$ . For technical reasons we will assume  $s \leq \frac{h}{1-\varepsilon}$ .

##### Unrestricted Adversary

Here we present the following negative result for unrestricted deterministic adversaries. Recall our assumption that  $\varepsilon$  is constant.

**Theorem 3.2.** *For every deterministic IGNORE algorithm ALG with slow speed  $s$ , and every  $h \geq s$ , there is a graph  $G$  with hamiltonicity  $h$  such that  $C_t^{\text{ALG}} = \Omega\left(\frac{h}{s}\right)$ .*

In order to prove the theorem we establish the following lemma for a special class of IGNORE algorithms. For these algorithms, which we call STARTATONCE, we make an agreement about when to start the first phase. General IGNORE algorithms wait an arbitrary (but deterministic) time before starting a phase based on the requests known so far. For STARTATONCE algorithms we demand that they start the first phase immediately at time  $t = 1$ . So in the first phase only the first request will be served.

**Lemma 3.3.** *Let  $s \leq 1$  and let ALG be any STARTATONCE algorithm that operates on the line on  $\{0, 1, \dots, n\}$ . Then there is a request sequence  $R$  with  $\text{ALG}(R) = \Omega\left(\frac{n^2}{s^2}\right)$  and  $\text{OPT}(R) = O\left(\frac{n}{s^2}\right)$ .*

*Proof.* We construct a request sequence  $R$  and describe the behaviour of an algorithm  $\text{ALG}'$ . Observe that the cost of serving  $R$  optimally can only be smaller than the cost of  $\text{ALG}'$ .

The general structure of our sequence is  $R = (n, 1, \dots, 1, 2, \dots, 2, \dots, n, \dots, n)$ . Initially, the server is at vertex 0. The idea is that the first request “fools” the STARTATONCE algorithm, which is then always “behind.”

A block of requests to a vertex  $i$  consists of  $p_i$  requests, where the  $p_i$  are determined as follows. For every vertex  $i = 1, \dots, n$ , moving from vertex  $i - 1$  to  $i$  requires time  $\frac{1}{s}$ . In the meantime  $\frac{1}{s}$  requests have arrived at  $i$  and  $\text{ALG}'$  starts serving those, where time  $\varepsilon$  is required for each. Meanwhile further requests arrive at  $i$  at rate 1. Since  $\varepsilon < 1$ , at some point in time all requests that queued up are served and  $\text{ALG}'$  would have to wait for new requests to arrive at  $i$ . We choose  $p_i$  such that at the  $p_i$ -st request to vertex  $i$  is the last one that can be served without having to wait for the next to arrive. Instead, the next request arrives at vertex  $i + 1$ . Hence the  $p_i$  have the property that  $\frac{1}{s} + p_i\varepsilon \geq p_i$  and thus  $p_i \leq \frac{1}{s(1-\varepsilon)}$ . We continue until vertex  $n$  is reached, where also the first request is served. Observe that the flowtime of each request to vertex  $i$  is at most  $\frac{1}{s} + \varepsilon \leq \frac{2}{s}$ , because the first request of a phase has longest waiting time; and where we have used  $\varepsilon < 1 \leq \frac{1}{s}$ .

Let  $p = \sum_i p_i + 1$  be the length of the sequence  $R$ . Since  $\text{ALG}'$  never waits,  $p$  is the smallest integer such that  $\frac{n}{s} + p\varepsilon \leq p$ , i.e.,  $p = \lceil \frac{n}{s(1-\varepsilon)} \rceil$ . We find  $\text{ALG}'(R) \leq \frac{2}{s}p + p + \varepsilon = O\left(\frac{n}{s^2}\right)$  assuming  $\varepsilon$  is constant.

By definition of STARTATONCE algorithms, ALG moves directly to position  $n$  which takes time  $\frac{n}{s}$ . Meanwhile  $\lfloor \frac{n}{s} \rfloor$  request arrive at certain vertices  $1, \dots, k$ , where  $k$  satisfies  $\frac{k}{s} + \lfloor \frac{n}{s} \rfloor \varepsilon \leq \frac{n}{s}$ . Hence  $k \leq n(1 - \varepsilon)$ . The flowtime of each of these  $\lfloor \frac{n}{s} \rfloor$  requests is at least  $\frac{n\varepsilon}{s}$  since this time is required for reaching these. Therefore  $\text{ALG}(R) \geq \lfloor \frac{n}{s} \rfloor \frac{n\varepsilon}{s} = \Omega\left(\frac{n^2}{s^2}\right)$ , again assuming that  $\varepsilon$  is constant.  $\blacksquare$

*Proof of Theorem 3.2.* Let  $h \geq s$  be given. We consider the line on the  $n = \lceil \frac{h}{s} \rceil + 1$  vertices  $V = \{0, 1, \dots, \lceil \frac{h}{s} \rceil\}$ , where the weight of each edge  $\{i - 1, i\}$  ( $i = 1, \dots, n$ ) is equal to  $s$ . Hence scaling by the factor  $s$  yields a graph that satisfies the assumptions of Lemma 3.3. This states that  $\text{OPT}(R) = O\left(\frac{n}{s^2}\right)$  and  $\text{ALG}(R) = \Omega\left(\frac{n^2}{s^2}\right)$ , which implies  $C_t^{\text{ALG}} = \Omega(n) = \Omega\left(\frac{h}{s}\right)$  for STARTATONCE algorithms.

In general, IGNORE algorithms can wait arbitrarily long before starting phase one. We modify the request sequence  $R$  used in the proof of Lemma 3.3 while maintaining the bounds hence proving the theorem. Note that for every deterministic IGNORE algorithm ALG, there is a unique time  $t$  when it decides to start its first phase given  $R$  as input.

If  $t \geq \lceil \frac{n}{s(1-\varepsilon)} \rceil$  OPT can serve the whole sequence in time  $\text{OPT}(R) = O\left(\frac{n}{s^2}\right)$  before ALG even starts. As at least  $\frac{n}{2s(1-\varepsilon)}$  requests have to wait time at least  $\frac{n}{s(1-\varepsilon)}$  we have  $\text{ALG}(R) = \Omega\left(\frac{n^2}{s^2}\right)$ .

Otherwise, let  $p < n$  be the position where the optimal server is located at time  $t$ . We define a sequence  $R'$  as follows. The first  $t$  requests of  $R$  and  $R'$  are identical. Then we append  $\lceil \frac{n}{2s} \rceil$  requests to position  $p$  and the remaining requests of  $R$ . Clearly  $\text{OPT}(R') \leq \text{ALG}'(R') = \text{ALG}'(R) + \lceil \frac{n}{2s} \rceil (1 + \varepsilon) = O\left(\frac{n}{s^2}\right)$  for serving the inserted requests and the additional flowtime of the first request.

At time  $t$  ALG starts its first phase (which contains the request to vertex  $n$ ) and requests are released at vertex  $p$ . After these  $\frac{n}{2s}$  requests are released ALG is at a position  $q \leq \frac{n}{2}$  and still has to serve  $r_1$  at vertex  $n$ . Hence the flowtime of each of the requests at  $p$  is at least  $\frac{n-q}{s} + \frac{n-p}{s} \geq \frac{n}{2s}$  since the server must reach  $p$  from  $q$  via  $n$ . Because there are  $\frac{n}{2s}$  requests at vertex  $p$ , we have  $\text{ALG}(R') = \Omega\left(\frac{n^2}{s^2}\right)$ .  $\blacksquare$

### Restricted Adversary

Now we define a restricted adversary which we will consider throughout this section. Before, we need additional notation.

Let  $r$  be a non-negative real and let  $i \in V$  be a vertex. The set  $B_i = \{j \in V : d_{i,j} \leq r\}$  is called the  $r$ -ball around  $j$ .

**Definition 3.4.** Let  $r$  be a non-negative real and let  $m$  be a positive integer. A request sequence  $R$  is called  $(r, m)$ -diffuse if for every vertex  $i \in V$  there are at least  $m$  requests in the set  $V - B_i$ , where  $B_i$  denotes the  $r$ -ball around  $i$ .

Let  $\ell$  be any positive integer and let  $R$  be a request sequence of length  $t$ . The  $\ell$ -partition of  $R$  is defined by  $R_1, R_2, \dots$ , where  $R_i = (r_{(i-1)\ell+1}, \dots, r_{i\ell})$ ; except for possibly the last  $R_i$  (which has  $t \bmod \ell$  elements). The  $R_i$  are also called  $\ell$ -phases of  $R$ .

**Definition 3.5.** Let  $r$  be a non-negative real, let  $m$  be a positive integer, and let  $\ell = \lceil \frac{r}{2s} \rceil$ . For any request sequence  $R$  let its  $\ell$ -partition be denoted  $R_1, R_2, \dots, R_p$ . The  $(r, m)$ -restricted adversary is allowed to choose any request sequence  $R$  such that every  $R_i$  in its  $\ell$ -partition is  $(r, m)$ -diffuse.

In the sequel we consider a certain strategy to handle the IGNORE-phases which we call WAIT. Initially, at time zero, both memories  $M$  and  $M'$  are empty and let  $w$  be a positive integer – to be specified later on. The strategy waits until exactly  $w$  requests have arrived and serves them using a path of length at most  $h$ . (In this paper, we do not deal with computing such a path. Notice, however, that an Eulerian path on a minimum tree that spans the vertices of the requests approximates its length by a factor of at most 2). The total flowtime achieved over all phases with this strategy is denoted  $\text{WAIT}_w$ . We require that the parameter  $w$  has to be chosen such that the time needed to serve the  $w$  requests in that manner is at most  $w$ . We show below, that this is possible for any speed  $s$  – even slow.

All phases induced by the WAIT strategy have the length  $w$  (except for possibly the last one), inducing a  $w$ -partition on  $R$ . Notice that the algorithm can decide if it is in the last phase: if no request has arrived within one time-unit, then no more requests will arrive. This fact avoids that the algorithm waits forever.

**Observation 3.6.** Let  $G$  be a graph with hamiltonicity  $h$ . For any slow speed  $s > 0$  and any service-time  $0 < \varepsilon < 1$  any integer  $w \geq \lceil \frac{h}{s(1-\varepsilon)} \rceil$  guarantees that the time needed to serve a  $\text{WAIT}_w$ -phase is at most  $w$ .

*Proof.* It suffices to find an integer solution of the inequality  $\frac{h}{s} + w\varepsilon \leq w$ . To see this observe that the expression  $\frac{h}{s}$  is an upper bound on the time the server spends in transit. Furthermore the server spends  $w\varepsilon$  time to serve all  $w$  pending requests. Therefore we have that  $w$  must satisfy  $w \geq \frac{h}{s(1-\varepsilon)}$ . Rounding to the next integer proves the claim.  $\blacksquare$

**Theorem 3.7.** *Let  $w = \lceil \frac{h}{s(1-\varepsilon)} \rceil$  be the parameter of the WAIT algorithm with a slow server. For any  $(r, m)$ -restricted adversary and sufficiently large  $t$  it holds that*

$$C_{(r, m)\text{-restricted}, t}^{\text{WAIT}_w} = O\left(\frac{h}{s} \cdot \frac{r+s}{rm}\right).$$

Hence  $C_{(r, m)\text{-restricted}, t}^{\text{WAIT}_w} = O(1)$  if the sequence satisfies  $m = \Omega\left(\frac{h}{s} \cdot \frac{r+s}{r}\right)$ .

**Lemma 3.8.** *For any request sequence  $R$  of length  $t$  and any integer  $w \geq \lceil \frac{h}{s(1-\varepsilon)} \rceil$  the strategy WAIT yields  $\text{WAIT}_w(R) \leq 2tw$ .*

*Proof.* We show that each request has flowtime at most  $2w$ . Observation 3.6 states  $w$  time units until the previous phase (during which the request has arrived) ends and at most  $w$  time units to serve the request in the current phase. Consequently we obtain  $\text{WAIT}_w(R) = \sum_i f_i \leq 2tw$  as claimed.  $\blacksquare$

**Lemma 3.9.** *Let  $r$  be a non-negative real,  $\ell = \lceil \frac{r}{2s} \rceil$ , and let  $m$  be a positive integer that may depend on  $r$ . Let  $i$  be an arbitrary initial position of the server (with speed  $s$ ) and let  $B_i$  be the  $r$ -ball around  $i$ . Let  $R$  be a request sequence of length  $\ell$  with  $m$  requests outside  $B_i$ , i.e., in the set  $V - B_i$ , then  $\text{OPT}(R) \geq \frac{mr}{2s}$ .*

*Proof.* Let  $r_1, \dots, r_m$  denote the requests outside  $B_i$  in the ordering in which the optimum solution serves them and let  $a_1, \dots, a_m$  be the times when these arrive. Furthermore let  $h_1, \dots, h_m$  denote the times immediately before these requests receive service.

Observe that we can write  $f_i = h_i - a_i + \varepsilon$  for every request  $r_i$ . Since the requests are served in the ordering  $r_1, \dots, r_m$  there are  $\delta_i \geq 0$  such that  $h_2 = h_1 + \varepsilon + \delta_1 \geq h_1 + \varepsilon$ ,  $h_3 = h_2 + \varepsilon + \delta_2 \geq h_1 + 2\varepsilon$ ,  $\dots$ ,  $h_i \geq h_1 + i\varepsilon$ . Consequently  $\text{OPT}(R) = \sum_j f_j \geq \sum_{j \in V - B_i} f_j \geq mh_1 - \sum_{j \in V - B_i} a_j$ .

Of the  $\ell$  requests that arrive during  $R$ ,  $m$  appear outside the  $r$ -ball  $B_i$ . This means that even in the optimum solution, the server has to travel distance at least  $r$  in order to reach (one of) those requests. In other words, the time to reach the boundary of  $B_i$  is at least  $\frac{r}{s}$  since the server travels at speed  $s$ . Hence  $h_1 \geq \frac{r}{s}$ .

Furthermore we have to bound  $\sum_{j \in V - B_i} a_j$  from above. We can rename the arrival times  $a_j$  into  $b_j$ , such that  $b_1 < b_2 < \dots < b_m$ . As there is one arrival per time unit, we have that the latest point in time of  $b_j$  is  $\ell - m + j - 1$ . Hence the choice  $\ell = \lceil \frac{r}{2s} \rceil$ ,  $m \geq 1$ , and  $\lceil \frac{r}{2s} \rceil \leq \frac{r}{2s} + 1$  imply

$$\sum_{j \in V - B_i} a_j = \sum_{j \in V - B_i} b_j \leq m\ell - m^2 + \frac{1}{2}m(m+1) - m = m\ell - \frac{1}{2}m(m+1) \leq m \left\lceil \frac{r}{2s} \right\rceil - m \leq \frac{mr}{2s}.$$

This bound and  $h_1 \geq \frac{r}{s}$  imply the result.  $\blacksquare$

*Proof of Theorem 3.7.* Lemma 3.2 and  $s \leq \frac{h}{1-\varepsilon}$  imply that  $\text{WAIT}_w(R) \leq 2tw \leq \frac{4th}{s(1-\varepsilon)}$ . Let  $p$  denote the number of complete  $\ell$ -phases; clearly  $p = \lfloor \frac{t}{\ell} \rfloor$ . Lemma 3.9,  $\ell \leq \frac{r}{2s} + 1$ , and  $t$  sufficiently large imply  $\text{OPT}(R) \geq \frac{mr}{2s}p \geq \frac{mr}{2s} \left(\frac{t}{\ell} - 1\right) \geq \frac{mr}{2s} \left(\frac{2ts}{r+2s} - 1\right) \geq \frac{mrt}{r+s}$ . With these bounds the result follows immediately.  $\blacksquare$

## 4 Randomized Adversaries

In this section, we define a randomized adversary, where we assume that the server is slow (since fast servers handle any sequence almost optimally). First recall the definitions of a  $(r, m)$ -diffuse sequence and the  $\ell$ -partition of a sequence.

The motivation behind the following definition is that we would like to relax the restrictions of a  $(r, m)$ -adversary. In specific, we only require that an  $\ell$ -phase is  $(r, m)$ -diffuse with a certain probability  $p$ , say. With this relaxation in mind, we ask how the expected competitive ratio behaves depending on  $p$ .

**Definition 4.1.** Let  $r$  be a non-negative real, let  $m$  be a positive integer, let  $\ell = \lceil \frac{r}{2s} \rceil$ , and let  $0 \leq p \leq 1$ . For any request sequence  $R$  let its  $\ell$ -partition be denoted  $R_1, R_2, \dots$ . A distribution on the set of all request sequences  $R$  of length  $t$  is called  $(r, m, p)$ -diffuse if  $\Pr[R_i \text{ is } (r, m)\text{-diffuse}] \geq p$  independently for every  $i$ . The  $(r, m, p)$ -restricted adversary is allowed to choose any  $(r, m, p)$ -diffuse distribution.

The following result relates deterministic and randomized  $(r, m)$ -adversaries. It states that the expected performance of WAIT against a randomized adversary is (up to a constant factor) equal to its performance against a deterministic adversary times the reciprocal of  $p$ .

**Theorem 4.2.** Let  $w = \lceil \frac{h}{s(1-\varepsilon)} \rceil$  be the parameter of the WAIT algorithm with a slow server. For any  $(r, m, p)$ -restricted adversary and sufficiently large  $t$  it holds that

$$E_{(r, m, p)\text{-restricted}, t}^{\text{WAIT}_w} = O\left(p^{-1} C_{(r, m)\text{-restricted}, t}^{\text{WAIT}_w}\right).$$

The following lemma relates  $\mathbb{E}\left[\frac{\text{ALG}}{\text{OPT}}\right]$  with  $\frac{\mathbb{E}[\text{ALG}]}{\mathbb{E}[\text{OPT}]}$ . The intended application is as follows. Suppose that there is a deterministic, worst-case upper bound  $w$  on the fraction  $\frac{\text{ALG}}{\text{OPT}}$  and that we can partition the probability space into two parts. The first part is a “good event”, which, whenever it occurs, implies that OPT is around its expectation. The second part is a “bad event” for which the value of  $\frac{\text{ALG}}{\text{OPT}}$  might be as large as  $w$ . However, the hope is that this bad event is sufficiently unlikely.

**Lemma 4.3.** Let  $X$  and  $Y$  be two positive random variables. Then for every  $\delta < 1$  we have

$$\mathbb{E}\left[\frac{X}{Y}\right] \leq \frac{\mathbb{E}[X]}{\mathbb{E}[Y](1-\delta)} + \mathbb{E}\left[\frac{X}{Y} \mid Y < \mathbb{E}[Y](1-\delta)\right] \Pr[Y < \mathbb{E}[Y](1-\delta)].$$

*Proof.* Condition on  $Y \geq \mathbb{E}[Y](1-\delta)$  and use  $\mathbb{E}[X \mid A] = \frac{\mathbb{E}[X] - \mathbb{E}[X \mid \bar{A}]\Pr[\bar{A}]}{\Pr[A]} \leq \frac{\mathbb{E}[X]}{\Pr[A]}$  which is true for non-negative random variables.  $\blacksquare$

*Proof of Theorem 4.2.* Our strategy is to apply Lemma 4.3 as described above. To that end, we have to estimate  $\mathbb{E}[\text{WAIT}_w]$ ,  $\mathbb{E}[\text{OPT}] - t$  (for an appropriate  $t$ ), a worst-case bound on  $\frac{\text{WAIT}_w}{\text{OPT}}$  and the probability that OPT deviates more than  $t$  from its expectation.

Firstly, since every request requires service-time  $\varepsilon$ , we have the lower bound  $\text{OPT}(R) \geq t\varepsilon$  for every request sequence  $R$  of length  $t$ . Hence, with Lemma 3.2 and  $s \leq \frac{h}{1-\varepsilon}$  we have  $\frac{\text{WAIT}_w}{\text{OPT}} \leq \frac{2w}{\varepsilon} \leq \frac{4h}{s(1-\varepsilon)}$ , which will serve as a worst-case upper bound. Recall that Lemma 3.2 states that we can bound  $\mathbb{E}[\text{WAIT}_w] \leq 2tw$  pessimistically from above.

Let  $D$  be a  $(r, m, p)$ -diffuse distribution on the request sequences of length  $t$ , say. Choose  $\ell = \lceil \frac{r}{2s} \rceil$  and consider the (random)  $\ell$ -partition  $R_1, R_2, \dots$  of the random request sequence  $R$ . For every phase  $R_i$  let  $M_i$  be the number of requests outside  $B_{j_i}$ , where  $B_{j_i}$  is the  $r$ -ball around the position  $j_i$  of the optimum server at the beginning of phase  $R_i$ . Furthermore define the indicator variable  $X_i$  for the event  $M_i \geq m$ . By the definition of  $(r, m, p)$ -diffuse distributions, the  $M_i$  (and hence also the  $X_i$ ) are independent. Further let  $X = \sum_i X_i$  and let  $\lfloor \frac{t}{\ell} \rfloor$  denote the number of (complete) phases. Lemma 3.9 implies that for every (complete) phase  $R_i$   $\text{OPT}(R_i) \geq \frac{rM_i}{2s} \geq \frac{mrX_i}{2s}$ .

Let us assume that  $t$  is sufficiently large for  $\lfloor \frac{t}{\ell} \rfloor \geq \frac{t}{2\ell}$ . Hence  $\lfloor \frac{t}{\ell} \rfloor \geq \frac{t}{2(\frac{r}{2s}+1)} = \frac{ts}{r+2s}$ . As  $\Pr[M_i \geq m] \geq p$  implies  $\Pr[X_i = 1] \geq p$  we have  $\mathbb{E}[\text{OPT}(R_i)] \geq \frac{mrp}{2s}$  and hence  $\mathbb{E}[\text{OPT}(R)] \geq \frac{mrp\lfloor \frac{t}{\ell} \rfloor}{2s} \geq \frac{mrpt}{2(r+2s)}$ .

Chernoff's inequality gives concentration of  $X$  around its mean:  $\Pr[X < \mathbb{E}[X](1 - \delta)] \leq e^{-\frac{\mathbb{E}[X]\delta^2}{2}}$ . Choose  $\delta = \frac{1}{2}$  and apply Lemma 4.3 using  $\mathbb{E}[X] \geq \frac{pt}{2} \frac{s}{r+s}$ . With  $t$  sufficiently large, i.e.,  $t = \omega\left(\log(h) \frac{r+s}{ps}\right)$  the result follows.  $\blacksquare$

## 5 An Optimal Phase-Strategy for the Line

In this section we prove that the optimal schedule of an IGNORE-phase can be computed in polynomial time for the special case when the underlying graph  $G$  is a line. A *line* is a graph  $G = (V, E)$  with vertices  $V = \{1, 2, \dots, n\}$  and edges  $E = \{\{1, 2\}, \{2, 3\}, \dots, \{n-1, n\}\}$  with edge-weights  $w(e) \geq 0$ .

Initially the server is at a certain vertex  $o \in V$ , called the *origin*. Any vertex  $i$  with  $i < o$  is to the *left*, any other vertex  $i \geq o$  is to the *right*. We write  $L = \{1, 2, \dots, o-1\}$  and  $R = \{o, o+1, \dots, n\}$ . We assume that for any vertex  $i \neq o$ , there are initially  $\varrho(i) > 0$  requests pending. Notice that the assumption that every vertex requires service is without loss of generality since we can always delete vertices  $i$  with  $\varrho(i) = 0$  and adjust the edge-weight between its former neighbours.

Because all requests in  $M$  are known at the beginning of each phase, computing the minimum total flow time  $F(M) = \sum_{r_i \in M} f_i$  is equivalent to finding a schedule with minimum total completion time  $C(M) = \sum_{r_i \in M} d_i$  (as the arrival times  $a_i$  are fixed). In the sequel, we show how to find such a schedule – represented by a permutation of the requests – with minimum total completion time.

But before, we give an example that shows that starting from the origin, serving all requests to the right and then all to the left (or the other way around) does not necessarily give an optimal schedule. Perhaps this is contrary to first intuition.

**Example 5.1.** Consider the line  $G = (V, E)$  on 4 vertices with edge weights  $w(\{1, 2\}) = x > 0$ ,  $w(\{2, 3\}) = 1$ ,  $w(\{3, 4\}) = y > 0$ , and respective numbers of requests  $\varrho(1) = a > 0$ ,  $\varrho(2) = 0$ ,  $\varrho(3) = 1$ , and  $\varrho(4) = b > 0$ . The initial vertex of the unit-speed server is 2. Consider the schedules 2314, 2134, and 2341. The first schedule serves in a zig-zag manner, the others from extreme point to extreme point. There are values for  $a, b, x, y$  such that 2314 has strictly less completion time than the others: A simple calculation gives  $C(2134) - C(2314) = 2x - 2b - a$  and  $C(2341) - C(2314) = 2ay + a - 2bx - 2b$ . It is of course possible to choose  $x$  and  $y$  sufficiently large such that both expressions are positive, which gives that the zig-zag-tour improves upon the extreme-point-tours.

**Theorem 5.2.** *An optimal schedule of an IGNORE-phase on a line  $G$  with  $n$  vertices can be computed by solving a shortest path problem on an appropriate graph on  $O(n^2)$  vertices.*

We introduce additional notation. For any subset  $U \subseteq V$  we denote  $\varrho(U) = \sum_{u \in U} \varrho(u)$ . For two vertices  $i$  and  $j$  we denote the total weight of the connecting shortest path by  $w(i, j) = w(\{i, i+1\}) + w(\{i+1, i+2\}) + \dots + w(\{j-1, j\})$ .

As a first step, we give Lemma 5.3 and Lemma 5.4 that establish that there is an optimal schedule which is *zig-zag* tour alternating between vertices to the left and the right of the origin  $o$ .

Since all requests are known at the outset, an optimal schedule does not involve idle time, i.e., the server is always either in transit or serves a vertex, but never waits. Hence an optimal schedule can be represented by a permutation of the requests. Lemma 5.3 even strengthens this observation and allows us to represent (optimal) schedules as permutations of the vertices  $1, 2, \dots, n$  only, rather than requests. Such a permutation is interpreted that the vertices are visited in the respective order, where all  $\varrho(i)$  pending requests are served upon such a visit.

**Lemma 5.3.** *There is an optimal schedule that serves all  $\varrho(i)$  requests at its first visit to  $i$ .*

*Proof.* We give an exchange argument. Consider a schedule  $S$  represented by a permutation of requests,  $S = (r_1, \dots, r_p)$ , say. Let  $r_i$  and  $r_j$  be two visits to the same vertex that are separated within  $S$ , i.e.,  $i+1 < j$ . Construct a schedule  $S'$  by moving  $r_j$  to position  $i+1$  pushing the  $d = j - i - 1$  requests  $r_{i+1}, \dots, r_{j-1}$  back:  $S' = (r_1, \dots, r_i, r_j, r_{i+1}, \dots, r_{j-1}, r_{j+1}, \dots, r_p)$ . The completion time of these requests hence increases by  $\varepsilon$ , each. However, the completion time of  $r_j$  decreases by  $d\varepsilon$  and a certain non-negative time the server is in transit returning to the vertex of  $r_j$ . Here we have used that all edge-weights are non-negative. The completion times of  $r_1, \dots, r_i$  do not change and those of  $r_{j+1}, \dots, r_p$  might even decrease for the same reason as before. Hence, the total completion time of the schedule  $S'$  does not increase over  $S$ . ■

**Lemma 5.4.** *Let  $r > o$  be a vertex to the right of the origin  $o$ . Then there exists an optimal schedule that serves vertex  $r-1$  before  $r$ . Analogously for a vertex  $\ell$  to the left which is visited before  $\ell-1$ . Hence there is an optimal schedule that contains the sequences  $(o, o+1, \dots, n)$  and  $(o-1, o-2, \dots, 1)$  as subsequences.*

*Proof.* We show the result for the vertices to the right; the other case is analogous. In specific, we prove that there is an optimal schedule that visits vertex  $n$  after vertex  $n-1$ . The general case then follows inductively.

Consider an optimal schedule  $S$  that serves vertex  $n$  before  $n-1$  and let  $B$  be the (possibly empty) set of vertices that are served between  $n$  and  $n-1$  in the schedule. As the server starts in  $o$ , it has to pass  $n-1$  in order to reach  $n$ . Consider the modified schedule  $S'$  in which  $n-1$  is served immediately before  $n$  is served leaving the remainder of the schedule unchanged. In  $S'$  every request in vertex  $n$  and in  $B$  is served  $\varrho(n-1)\varepsilon$  time units later than in  $S$ , which yields an increase in total completion time of  $(\varrho(n) + \varrho(B))\varrho(n-1)\varepsilon$ . Visiting  $n-1$  earlier yields a decrease of  $(\varrho(n) + \varrho(B))\varepsilon$  for every of the  $\varrho(n-1)$  requests there and possibly a gain due to transit time savings. This change decreases the total completion time of vertex  $n-1$  by at least  $(\varrho(n) + \varrho(B))\varrho(n-1)\varepsilon$ . Also for the reason of possible transit time savings, the completion times of the requests served after vertex  $n-1$  in the schedule  $S$  can only decrease in  $S'$ . This yields that  $S'$  is an optimal schedule that serves  $n-1$  before  $n$ . ■

*Proof of Theorem 5.2.* We show that an optimal permutation of the vertices can be found by solving a shortest path problem on the following graph  $G$ . Without loss of generality, we assume that the server has speed equal to one, because we can always adjust the edge weights of the line accordingly.

In the graph  $G$ , there is one distinguished vertex denoted  $(o, o)$  which is associated to the origin  $o$  of the line and the initial vertex of the single-source shortest path problem. The other vertices of  $G$  are labeled  $(i, j)$ , where  $i$  and  $j$  are any two vertices of the line that are on opposite sides of the origin  $o$ , i.e.,  $i \in L$  and  $j \in R$  or  $i \in R$  and  $j \in L$ . The edges of the graph correspond to possible segments of a zig-zag tour. Any two vertices  $(i, j)$  and  $(j, k)$  with  $j \in L$  and  $i < k \in R$  or  $j \in R$  and  $k < i \in L$  are connected by (directed) edges.

Each such edge, denoted  $(i, j, k)$  has cost  $c(i, j, k)$  which we describe in the sequel. These cost correspond to the increase in total completion time when the server visits the previously unvisited vertex  $k$  starting from vertex  $j$ , where  $i$  is the vertex closest to  $k$  which has already been visited. Hence the vertices  $S = \{i + 1, \dots, k\}$  are unvisited yet but will be visited while the server is on its way to  $k$ . For any vertex  $\ell \in S$  the time until all its requests are served is  $c(\ell) = w(j, \ell) + \varrho(\{i + 1, \dots, \ell\})\varepsilon$ ; write  $c(S) = \sum_{\ell \in S} c(\ell)$ . The other vertices  $U = \{1, \dots, j-1\} \cup \{k+1, \dots, n\}$  remain unserved so far and their total increase in completion time is  $c(U) = \varrho(U)(w(j, k) + \varrho(S)\varepsilon)$ . We define that the cost of an edge  $(i, j, k)$  is  $c(i, j, k) = c(U) + c(S)$ .

Clearly, any path that starts at  $(o, o)$  and ends up  $(1, n)$  or  $(n, 1)$  corresponds to a zig-zag tour that visits all vertices on the line; and vice versa. Hence, by Lemma 5.4, finding a shortest such path gives rise to an optimal schedule.  $\blacksquare$

## References

- [1] AUSIELLO, G., ALLULLI, L., BONIFACI, V., AND LAURA, L. On-line algorithms, real time, the virtue of laziness, and the power of clairvoyance. *3rd International Conference on Theory and Applications of Models of Computation, (TAMC '06)* (2006), 1 – 20.
- [2] AUSIELLO, G., FEUERSTEIN, E., LEONARDI, S., STOUGIE, L., AND TALAMO, M. Competitive algorithms for the travelling salesman. *4th Workshop on Algorithms and Data Structures (WADS '95)* (1995), 206 – 217.
- [3] BLOM, M., KRUMKE, S. O., DE PAEPE, W., AND STOUGIE, L. The online-TSP against fair adversaries. *4th Italian Conference of Algorithms and Complexity (CIAC '00)* (2000), 137 – 149.
- [4] GUTIÉRREZ, S., KRUMKE, S. O., MEGOW, N., AND VREDEVELD, T. How to whack moles. *1st Workshop on Approximation and Online Algorithms (WAOA '03)* (2003), 192 – 205.
- [5] HAUPTMEIER, D., KRUMKE, S. O., AND RAMBAU, J. The online dial-a-ride problem under reasonable load. *4th Italian Conference of Algorithms and Complexity (CIAC '00)* (2000), 125–136.
- [6] KOUTSOPIAS, E., AND PAPADIMITRIOU, C. Beyond competitive analysis. *35th Annual Symposium on Foundations of Computer Science (FOCS '94)* (1994), 394 – 400.

- [7] PANAGIOTOU, K., AND SOUZA, A. On adequate performance measures for paging. *38th Annual ACM Symposium on Theory of Computing (STOC '06)* (2006).
- [8] SHMOYS, D. B., WEIN, J., AND WILLIAMSON, D. P. Scheduling Parallel Machines Online. *32nd Annual Symposium on Foundations of Computer Science (FOCS '91)* (1991), 131–140.
- [9] SLEATOR, D. D., AND TARJAN, R. E. Amortized efficiency of list update and paging rules. *Communications of the ACM* 28, 2 (1985), 202 – 208.