An adversarial queueing model for online server routing

Vincenzo Bonifaci*

Abstract

In an online server routing problem, a vehicle or *server* moves in a network in order to process incoming *requests* at the nodes. Online server routing problems have been thoroughly studied using competitive analysis. We propose a new model for online server routing, based on adversarial queueing theory. The model addresses questions such as whether a server routing algorithm is *stable*, that is, whether it is such that the number of unserved requests in the system remains bounded at all times, assuming a bound on the global rate of requests arrival. This captures a notion of throughput for which competitive analysis typically does not give any useful result. In this framework, we consider a number of natural algorithms and we analyze their stability and performance.

Keywords: online algorithms, adversarial queueing theory, server routing.

1 Introduction

Consider the following model of a computer harddrive: while the disk is rotating, read and write requests arrive for data lying on specific sectors of the disk. Thus, the head located on the arm of the disk has to move in order to align itself to the correct track, and wait for the disk to rotate onto the sector holding the data. If we ignore the rotational delay, the problem is that of routing a *server* (the head) on a finite space (the arm) in order to process some *requests*.

A sensible algorithm for controlling the disk's head should be able to cope with requests in a *stable* manner: the number of unserved requests should not increase indefinitely as time passes. This requires of course that the rate of incoming requests does not overflow the head's service speed, but that condition alone is not sufficient; an algorithm that moves the head back and forth between far away locations while serving few requests will easily lead to an unstable system even when the arrival rate is relatively low. Notice that this

*Dip. di Informatica e Sistemistica, Università di Roma "La Sapienza", Rome, Italy; Dept. Mathematics and Computer Science, Technical University Eindhoven, Eindhoven, the Netherlands. E-mail: bonifaci@dis.uniroma1.it. This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

stability requirement can be seen as requiring the algorithm to have an optimal *throughput*, since no algorithm with suboptimal throughput can keep the system stable if the load remains high for a sufficient amount of time. One would also like the average delay experienced by the requests to be as small as possible.

The optimization part of this problem could be seen as an *online server routing problem* [4, 8] with the *average flow time* objective. However, it is well-known from the online algorithm literature that there cannot be constant competitive algorithms for this problem. Thus, pure competitive analysis is unable to distinguish the behavior of different algorithms. Restrictions to the offline adversary and alternative models have been proposed that try to overcome this issue [6, 9].

In this work we propose a new approach that, while abandoning competitive analysis, still assumes worst case behavior of the inputs. The approach is based on *adversarial queueing theory*, which has been recently proposed to analyze online packet routing problems [2, 5]. We propose an adversarial queueing model specific for online server routing, and in this framework we analyze the stability and the performance of several natural algorithms. This allows us to easily distinguish between algorithms that would fare equally badly from the point of view of competitive analysis.

The rest of the paper is structured as follows. In Section 2, we discuss previous related work. The model itself is presented in Section 3. In Section 4 several natural algorithms are introduced and analyzed in terms of the model. Section 5 presents a lower bound on the number of unserved requests that explains why the upper bounds of Section 4 are essentially best possible. We end with some directions for further research.

2 Related work

The adversarial queueing theory framework was first formulated by Borodin et al. [5], who considered packet routing problems in networks with continuous packet arrivals. The model replaces the probabilistic assumptions usual in queueing theoretical analyses with worst case inputs. Thus the name *adversarial queueing theory* was proposed to stress that while the issue studied is that of stability – the crucial issue of queueing theory – the approach is of adversarial nature. Following the work of Borodin et al., Andrews et al. [2] considered several natural algorithms in this framework and gave many stability and instability results.

In the context of online server routing problems, the model that comes closer to the one we propose is the *reasonable load* model by Hauptmeier et al. [6]. Roughly speaking, they assume that every set of requests that come up in a sufficiently large time period can be served in a time period of at most the same length. Under this assumption, they consider the dial-a-ride problem for the minimization of maximum (or average) flow time, and they distinguish the behavior of two algorithms, Replan and Ignore, that would be indistinguishable by pure competitive analysis. We also give similar results for Replan, Ignore

and several other algorithms. Other results in related directions of research, albeit in quite different models, are given by Alborzi et al. [1] and Irani et al. [7].

Experimental results for different disk scheduling policies were given by Teorey and Pinkerton [10]. More recently, new algorithms for disk scheduling have been proposed by Andrews et al. [3].

3 The model

An *online server routing system* consists of a triple (G, A, P), where G is a graph, A is an adversary and P is an online algorithm. We now further detail each of these components.

The undirected, connected graph $G = (V, E)$ represents the space where requests are injected by the adversary and where the server operated by the online algorithm moves. A special vertex $o \in V$ is marked as the origin. We let n be the number of vertices of G and δ its diameter.

We use a discrete time model. At every time step, the server operated by the algorithm can either cross an edge or serve a single request from its current location (but not both). At time 0, the server is located at the origin.

We consider two types of adversaries, a stronger one used in all the positive results and a weaker one for the negative results; this difference can only strengthen the results. A *strong adversary* of rate $\lambda \in (0, 1]$ and burst $\mu > 0$ can, during any time interval I, inject at most $\lambda|I| + \mu$ requests overall anywhere on the vertices of the graph. A *weak adversary* of rate $\lambda \in (0, 1]$ can, during any time interval I, release at most $\lceil \lambda|I| \rceil$ requests overall anywhere on the vertices of the graph. The sequence generated by the adversary is denoted by $\sigma = \sigma_1\sigma_2 \cdots$. Every request σ_j is a pair $(r_j, x_j) \in \mathbb{Z}_+ \times V$, where r_j is the release date of the request (the time it becomes available) and x_j its location (a vertex of G). We denote by C_j the completion time of request σ_j, i.e., the time unit following the one during which the server started processing σ_j. If the request is never served by the algorithm, we conventionally define $C_j = \infty$.

More formally, the model can be described as follows. At every time step t, the current *configuration* $\mathcal{C}_t$ of the system is a vertex $s(t) \in V$ plus a collection of sets $\{U_v^t : v \in V\}$, such that $s(t)$ is the position of the server at time t and U_v^t is the set of requests waiting at v at the time t. From the configuration $\mathcal{C}_t$ we obtain the configuration $\mathcal{C}_{t+1}$ as follows. The adversary adds new requests to some of the sets U_v; then the algorithm either chooses $s(t+1)$ such that $\{s(t), s(t+1)\} \in E$ or it removes a request from $U_{s(t)}$ and leaves $s(t+1) = s(t)$. A *time-evolution* of G, of rate λ and burst μ, is a sequence of such configurations $\mathcal{C}_0, \mathcal{C}_1, \ldots$, such that for all intervals I, no more than $\lambda|I| + \mu$ requests are introduced during I in G. By the *system* (G, A, P) we mean the time-evolution of G induced by adversary A and algorithm P with initial configuration $s(0) = o$ and $U_v^0 = \emptyset$ for all $v \in V$.

Our results are centered around the following concepts.

Definition 3.1. An online server routing system (G, A, P) is *stable* if there exists a constant Q (which may depend on the system) such that

$$\sum_{v \in V} |U_v^t| \leq Q$$

for all $t \in \mathbb{Z}_+$, that is, the total number of unserved requests is bounded by Q at all times. Otherwise we say that the system is *unstable*.

Definition 3.2. An algorithm P is *universally stable* if for every graph G and every strong adversary A of rate λ and burst μ with $\lambda < 1$, the system (G, A, P) is stable.

Definition 3.3. Given a system (G, A, P), the *average flow time up to time t* of the system is

$$\bar{F}(t) = \frac{1}{N(t)} \sum_{j=1}^{N(t)} (\min(t, C_j) - r_j)$$

where $N(t)$ is the number of requests injected by the adversary up to time t. A system has *bounded average flow time* if there exists a constant F such that the average flow time up to time t is at most F for all t.

The connection between stability and average flow time is given by the following.

Proposition 3.1. *If a system (G, A, P) is stable and there exists $\eta \in (0, 1]$ such that $N(t) \geq \eta t$ for all $t \in \mathbb{Z}_+$, then the system has bounded average flow time.*

Proof. In the expression for the average flow time of the system up to time t, we can sum over time instead of summing over requests to obtain

$$\bar{F}(t) = \frac{1}{N(t)} \sum_{i=0}^{t-1} \sum_{v \in V} \left|U_v^i\right|.$$

Since the system is stable, $\sum_v \left|U_v^i\right| \leq Q$ for all i, thus $\bar{F}(t) \leq tQ/N(t) \leq Q/\eta$ and the system has bounded average flow time. □

4 Stability results

In this section we consider several natural algorithms for the online server routing model we have introduced, and we classify each of them according to stability. A summary of the results is presented in Table 1.

As a preliminary result, we remark that, in Definition 3.2, it is necessary to consider only adversaries with rate λ strictly less than 1, as otherwise for any algorithm the system can be unstable.

Proposition 4.1. *Let P be any algorithm and G any nontrivial graph. Then there exists an adversary A of rate 1 such that the system (G, A, P) is unstable.*

Algorithm	Univ. stable?	Reference
FIFO	no	Th. 4.3
SSTF	no	Th. 4.4
REPLAN	no	Th. 4.5
IGNORE	yes	Th. 4.6
TREE-SCAN	yes	Th. 4.7

Table 1: Universal stability of different server routing algorithms.

Proof. The adversary injects at every time step a request on a vertex that is distinct from the current location of the server. The server must change location infinitely often, but every time it does, the number of unserved requests increases by one. □

In the following, we assume without loss of generality that at every vertex of the graph there is a *queue* holding the requests pending at that vertex and that the algorithm always serves the oldest request of a queue.

A general term used in the algorithms is that of "emptying" the queue at a given vertex v of the graph. By this we mean that a request in v is served at every time step until no more requests in v are available. This process takes a finite amount of time because the rate of the adversary is strictly less than one. Moreover, some of the algorithms we consider are undefined in the case of ties. In those cases, we assume the worst tie-breaking rule for the positive results and the best for the negative ones.

The following technical lemma is useful when establishing instability results. Given two adversaries A_1, A_2, their *union* $A_1 \cup A_2$ is the adversary that releases the requests of both adversaries.

Lemma 4.2. *Let A_1 be a weak adversary of rate $\lambda \leq 1/2$. Then for every $v \in V$ and every $\epsilon \in (0, 1/6)$, there exists a weak adversary A_2 of rate ϵ releasing requests in v such that $A_1 \cup A_2$ is a weak adversary of rate at most $2/3 + \epsilon$.*

Proof. We define A_2 as follows: it releases requests in v only at time steps during which A_1 does not release any request, and so that its rate is ϵ; this is always possible by taking ϵ sufficiently small. Apart from these constraints, the precise release dates of the requests of A_2 are irrelevant for the lemma.

We have to prove that for any interval I of length t, the requests released by $A_1 \cup A_2$ are at most $\lceil (2/3 + \epsilon)t \rceil$.

Consider any such interval I. If $t = 1$, the claim holds simply because A_1 and A_2 never release a request during the same time step.

For all other t, notice that the number of requests released by A_1 (resp. A_2) is at most $\lceil \lambda t \rceil$ (resp. $\lceil \epsilon t \rceil$). We prove the claim by showing that $\lceil \lambda t \rceil + \lceil \epsilon t \rceil \leq \lceil (2/3 + \epsilon)t \rceil$. Notice that $\epsilon \leq 2/3 - \lambda$. When $t \geq 1/(2/3 - \lambda)$,

$$\lceil \lambda t \rceil + \lceil \epsilon t \rceil \leq \lceil \lambda t + 1 + \epsilon t \rceil \leq \lceil \lambda t + (2/3 - \lambda)t + \epsilon t \rceil \leq \lceil (2/3 + \epsilon)t \rceil .$$

When $t < 1/(2/3 - \lambda)$, notice that $\lceil \epsilon t \rceil = 1$. If t is even, say $t = 2q$,

$$\lceil \lambda t \rceil \leq \lceil t/2 \rceil = q \leq (4/3)q = (2/3)t.$$

If t is odd, $t = 2q + 1$ where $q \geq 1$,

$$\lceil \lambda t \rceil \leq \lceil t/2 \rceil = q + 1 \leq (4/3)q + 2/3 = (2/3)t.$$

Thus, in both cases,

$$\lceil \lambda t \rceil \quad \leq \quad (2/3)t < (2/3)t + \epsilon t$$

from which it follows

$$\lceil \lambda t \rceil + \lceil \epsilon t \rceil = \lceil \lambda t \rceil + 1 \quad \leq \quad \lceil (2/3)t + \epsilon t \rceil .$$

□

Algorithm 1 FIFO - First In First Out

The server processes the requests in the same order as their release dates.

In the context of single-machine scheduling, the FIFO algorithm (Algorithm 1) is optimal for the minimization of the maximum flow time of jobs. However, a similar approach fails in server routing because of the costs incurred by moving between distant requests.

Theorem 4.3. *For every nontrivial graph G, there is a weak adversary A such that (G, A, FIFO) is unstable.*

Proof. Consider any edge $\{v, w\}$ of the graph, and suppose that the server starts in w. Fix $\lambda \in (1/2, 1)$. The requests are given alternatively in v and w at rate λ. Note that FIFO serves requests at a rate of at most $1/2$, since it needs to move after serving each request. But $\lambda > 1/2$, thus the system is unstable. In general, if the graph has diameter δ, FIFO is unstable at every rate greater than $1/(\delta + 1)$. □

Another natural algorithm is SSTF or Shortest Seek Time First (Algorithm 2). The algorithm attempts to greedily minimize the distance traveled between the service of any two requests.

Algorithm 2 SSTF - Shortest Seek Time First

SSTF works in phases. At the beginning of each phase, let v be the vertex with a nonempty queue that is nearest to the current position of the server. At every step during the phase, the server moves along a shortest path from the current vertex to v. When it reaches v, it proceeds to empty v. When v has been emptied, the phase ends.

Theorem 4.4. *For every graph G of diameter at least 3 there is a weak adversary A such that (G, A, SSTF) is unstable.*

Proof. Consider any chain $v_0v_1v_2v_3$ of length 3 in G, and suppose that the server starts in v_1. A first adversary A_1 gives requests alternatively in v_0 and v_1, at a rate of $1/2$, so that as soon as a request is served in v_i a new request appears in v_{1-i}. By Lemma 4.2, for any $\epsilon > 0$ there exists an adversary A_2 of rate ϵ releasing requests in v_3 such that $A_1 \cup A_2$ has rate at most $2/3 + \epsilon$. Notice that requests of A_1 keep the server between v_0 and v_1, so that requests in v_3 will never be served. Thus the system $(G, A_1 \cup A_2, \text{SSTF})$ is unstable. □

Two classical strategies for generic online service are REPLAN and IGNORE (Algorithms 3 and 4). Although for the purpose of minimizing the average flow time they perform equally badly from the point of view of competitive analysis, the following results establish IGNORE as a more robust algorithm. These results are similar to those of Hauptmeier et al. [6].

Algorithm 3 REPLAN

REPLAN maintains a shortest walk on the set of vertices that have unserved requests. Whenever the current vertex is nonempty, REPLAN serves a request there. Otherwise, it moves the server along the shortest walk. Whenever a new request is released, the shortest walk is recomputed.

Algorithm 4 IGNORE

IGNORE works in phases. At the beginning of each phase, let o be the current position of IGNORE. IGNORE computes a shortest schedule on the set of currently unserved requests starting and ending at o. During the phase, IGNORE follows the schedule, ignoring temporarily requests released after the beginning of the phase. When the schedule has been completed, the phase ends.

Theorem 4.5. *For every graph G of diameter at least 3 there is a weak adversary A such that (G, A, REPLAN) is unstable.*

Proof. Consider any chain $v_0v_1v_2v_3$ of length 3 in the graph, and suppose that the server starts in v_0. The first adversary A_1 gives the following requests: $(0, v_3)$, $(3, v_0)$, $(6, v_3)$, and $(9 + 3i, v_0)$ for all $i \geq 0$. The sequence is such that starting from time 9, the server will never reach v_3 again. The rate of A_1 is $1/3$; thus, by Lemma 4.2, for any sufficiently small $\epsilon > 0$ there exists an adversary A_2 of rate ϵ releasing requests in v_3 such that $A_1 \cup A_2$ has rate at most $2/3 + \epsilon$. The requests released by A_2 do not change the behavior of REPLAN, since they are always scheduled after the requests released by A_1. Thus the requests of A_2 are never served and the system is unstable. □

Theorem 4.6. IGNORE *is universally stable.*

Proof. Consider any graph of n vertices and any strong adversary of rate $1-\epsilon$ and burst μ. We start by proving by induction that at the beginning of every phase the number of unserved request is bounded by

$$2(1-\epsilon)(n-1)/\epsilon + \mu/\epsilon.$$

Let $[t, t')$ be a phase, let z be the number of requests served during the phase and let u be the number of unserved requests at the beginning of the phase. Then, by definition of IGNORE,

$$t' - t \leq 2(n-1) + z \tag{1}$$

since when IGNORE does not serve a request, its server moves along a shortest closed walk spanning all the requests unserved at time t, and this walk cannot be longer than twice the number of edges of a spanning tree of the graph. By definition, IGNORE serves all those requests, which means that

$$z = u. \tag{2}$$

The number of requests unserved at time t' is then at most

$$\begin{array}{rcll} u + (1-\epsilon)(t'-t) + \mu - z & \leq & 2(1-\epsilon)(n-1) + \mu + u - \epsilon z & \text{by (1)} \\ & \leq & 2(1-\epsilon)(n-1) + \mu + (1-\epsilon)u & \text{by (2)} \\ & \leq & 2(1-\epsilon)(n-1)/\epsilon + \mu/\epsilon & \text{by induction.} \end{array}$$

To conclude the proof we need to show that the duration of each phase is bounded by a constant. But that is true because the duration of each phase is at most $2n + u$, where u is the number of unserved requests at the beginning of the phase, and we just proved that u is bounded. □

Finally, we propose an algorithm called TREE-SCAN (Algorithm 5) that can be seen as a generalization of the algorithm Scan frequently used in disk scheduling [10]. Notice that TREE-SCAN, differently from an algorithm such as IGNORE, does not use phases and is greedy, in the sense that it always serves a request from the current location of the server if possible.

Algorithm 5 TREE-SCAN

Let W be a closed Eulerian walk on the graph obtained by doubling all the edges of a spanning tree of G.
At every time step, if the current vertex has a nonempty queue, TREE-SCAN serves a request on the current vertex. Whenever the current vertex v has an empty queue, TREE-SCAN moves the server to the vertex following v in the walk W.

Theorem 4.7. TREE-SCAN *is universally stable.*

Proof. Again, consider any graph of n vertices and any strong adversary of rate $1-\epsilon$ and burst μ. The proof is by induction on time steps. Consider any time t'. Let t be the latest time before t' during which the server was located at $s(t')$ and such that between t and t' TREE-SCAN visited the whole graph (if there is no such t, let $t=0$). Let z be the number of requests served during $[t,t')$. Then

$$t'-t \leq z+2(n-1), \tag{3}$$

since at each time step either the algorithm serves a request or it moves to the next vertex in the walk W. Also, let u be the number of unserved requests at time t. The inductive hypothesis is that $u \leq 2(1-\epsilon)(n-1)/\epsilon + \mu/\epsilon$. All requests unserved at time t are served at time t' because the algorithm always serves requests when visiting a vertex and during the interval $[t,t')$ the server visited the entire graph; thus

$$z \geq u. \tag{4}$$

The number of requests unserved at time t' is then at most

$$\begin{array}{rcll} u+(1-\epsilon)(t'-t)+\mu-z & \leq & 2(1-\epsilon)(n-1)+\mu+u-\epsilon z & \text{by (3)} \\ & \leq & 2(1-\epsilon)(n-1)+\mu+(1-\epsilon)u & \text{by (4)} \\ & \leq & 2(1-\epsilon)(n-1)/\epsilon+\mu/\epsilon & \text{by induction.} \end{array}$$

□

5 A lower bound

In this section we show that, for every $\lambda > 1/2$, there is an adversary of rate λ that can force the number of outstanding requests to grow up to $\Theta(n)$ on any n-vertices graph, regardless of the algorithm used for service.

Theorem 5.1. *Let P be any algorithm and G a graph on n vertices. For every $\epsilon \in (0,1/2)$, there exists a weak adversary A of rate $1-\epsilon$ such that eventually the total number of unserved requests in the system (G,A,P) is at least*

$$(1-2\epsilon)(n-1)/\epsilon + 1.$$

Proof. We break the construction of A into phases. A *v-phase* starts when the server moves to $v \in V$ and ends when it moves to some other vertex. At time t in a v-phase, A injects a request at a vertex $w \in V \setminus \{v\}$ minimizing $|U_w^t|$, as long as this is compatible with the rate of A. We prove the claim by showing that in any phase, either the total number of unserved request increases, or there were already more than $(1-2\epsilon)(n-1)/\epsilon + 1$ requests pending.

Consider any v-phase. Let u_v, u_{-v} (u'_v, u'_{-v}, resp.) be the number of unserved requests at v and $V \setminus \{v\}$ at the start (end, resp.) of the phase. If z_v is the number of requests served during the phase at v, we have by construction

$$u'_v = u_v - z_v \tag{5}$$

$$u'_{-v} = u_{-v} + (1-\epsilon)(1+z_v) \tag{6}$$

$$z_v \leq u_v. \tag{7}$$

Suppose that the total number of unserved requests in the system does not increase, that is

$$u'_v + u'_{-v} - (u_v + u_{-v}) \leq 0. \tag{8}$$

Combining (8) with (5) and (6),

$$(1-\epsilon)(1+z_v) - z_v \leq 0.$$

Solving for z_v, we get

$$z_v \geq (1-\epsilon)/\epsilon$$

and by (7), we obtain

$$u_v \geq (1-\epsilon)/\epsilon$$

so that the number of unserved requests at v at the beginning of the phase is already at least $(1-\epsilon)/\epsilon$. But then consider the first time t that $|U_v^t| \geq (1-\epsilon)/\epsilon$. By the way the adversary is defined, this can happen only when $n-2$ other vertices contain at least $(1-\epsilon)/\epsilon - 1 = (1-2\epsilon)/\epsilon$ requests each. □

A consequence of Theorem 5.1 is that, when $\lambda > 1/2$, the upper bound on the number of outstanding requests for IGNORE and TREE-SCAN given in the proofs of Theorems 4.6 and 4.7 is best possible up to a factor that depends on λ but not on n.

6 Further directions

Our model suggests several open problems, as it can be easily generalized in a number of directions. For example we may consider the dial-a-ride problem [8], in which requests have both a source and a destination and the server is a vehicle with some capacity; this allows to study applications such as elevator scheduling. In this case, of course, the rate constraint on the adversary should be reformulated appropriately, as done in Hauptmeier et al. [6]. Also, in our model the processing time of a request is directly related to the speed of the server: serving a request takes the same time as moving to an adjacent vertex. It is natural to ask about the impact of arbitrary processing times.

Finally, in our opinion the most interesting problem suggested by our model is the following: consider the online server routing problem on a graph, where the objective is now *minimization of the maximum number of unserved requests at any time*. Can we find a competitive algorithm for this problem? The competitive ratio would necessarily depend on the characteristics of the graph (otherwise it is easy to prove that no such algorithm can exist). In any case, we think that such a competitive algorithm would be interesting because it would be able to maintain a near-optimal number of unserved requests not only under heavy load conditions (like IGNORE or TREE-SCAN), but also under light load.

References

[1] H. Alborzi, E. Torng, P. Uthaisombut, and S. Wagner. The k-client problem. *Journal of Algorithms*, 41(2):115–173, 2001.

[2] M. Andrews, B. Awerbuch, A. Fernández, T. Leighton, Z. Liu, and J. M. Kleinberg. Universal-stability results and performance bounds for greedy contention-resolution protocols. *Journal of the ACM*, 48(1):39–69, 2001.

[3] M. Andrews, M. A. Bender, and L. Zhang. New algorithms for disk scheduling. *Algorithmica*, 32(2):277–301, 2002.

[4] V. Bonifaci. *Models and Algorithms for Online Server Routing*. PhD thesis, Technical University Eindhoven, The Netherlands, 2007. Available at `http://www.dis.uniroma1.it/~bonifaci/papers/phdthesis-tue.pdf`.

[5] A. Borodin, J. M. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, 2001.

[6] D. Hauptmeier, S. O. Krumke, and J. Rambau. The online dial-a-ride problem under reasonable load. In G. Bongiovanni, G. Gambosi, and R. Petreschi, editors, *Proc. 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 2000.

[7] S. Irani, X. Lu, and A. Regan. On-line algorithms for the dynamic traveling repair problem. *Journal of Scheduling*, 7(3):243–258, 2004.

[8] S. O. Krumke. Online optimization: Competitive analysis and beyond. Habilitation Thesis, Technical University of Berlin, 2001.

[9] 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 K. Jansen, S. Leonardi, and V. V. Vazirani, editors, *Proc. 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pages 200–214. Springer-Verlag, 2002.

[10] T. J. Teorey and T. B. Pinkerton. A comparative analysis of disk scheduling policies. *Communications of the ACM*, 15(3):177–184, 1972.