Competitive Online Scheduling with Level of Service

Ee-Chien Chang *
Department of Computational Science
National University of Singapore

Chee Yap *
Department of Computer Science
Courant Institute, New York University

July 7, 1988 (Revised: Oct 13, 2000)

Abstract

Motivated by an application in thinwire visualization, we study an abstract on-line scheduling problem where the size of each requested service can be scaled down by the scheduler. Thus our problem embodies a notion of "Level of Service" that is increasingly important in multimedia applications. We give two schedulers `FirstFit` and `EndFit` based on two simple heuristics, and generalize them into a class of greedy schedulers. We show that both `FirstFit` and `EndFit` are 2-competitive, and any greedy scheduler is 3-competitive. These bounds are shown to be tight.

1 Introduction

We study an abstract on-line scheduling problem motivated by visualization across a "thinwire" network [4, 3]. An example of such a visualization problem is a server-client model where the server and client are connected by a thinwire (that is, a bandwidth-limited connection such as the Internet), with the the server holding a very large image that the client wishes to visualize. The viewer on the client side can control the transmission process by moving a mouse cursor over a low-resolution copy of the image to be visualized. This mouse motion generates, in real-time, a sequence of sampled positions along the mouse cursor trajectory. Each sampled position (x, y) corresponds to a request for higher resolution data at the position. As the bandwidth is limited, we

*Partially funded by NSF grant CCR-9619846

could only partially serve each request. This is where an on-line scheduler is needed to optimize the decisions. In most scheduling problems, a partially served request does not contribute to the performance of the scheduler. However, in this problem, a partially sent data can still provide useful information to the user. Thus, instead of sending all the requested data, the server has the option of lowering the "level" of the requested service in order to gain an overall better response time. This paper focuses on this level of service property. Note that there is considerable interest in similar Quality of Service (QoS) issues in multimedia research.

We use the standard notion of "competitiveness" in the sense of Sleator and Tarjan [9] to judge the quality of our online schedulers. A *scheduler* S produces a feasible schedule $S(I)$ for each instance I of our scheduling problem. Each $S(I)$ has an associated *merit*, where $\mathit{merit}(S(I)) \geq 0$. Let $\mathtt{opt}(I)$ denote any feasible schedule for I that maximizes the merit. We say S is *c-competitive* ($c \geq 1$) if for all I,

$$\mathit{merit}(\mathtt{opt}(I)) \leq c \cdot \mathit{merit}(S(I)) + b,$$

where b is a fixed constant. The *competitive ratio* of S is defined by

$$C(S) := \sup_I \frac{\mathit{merit}(\mathtt{opt}(I))}{\mathit{merit}(S(I))}.$$

Thus, we want schedulers S with $C(S) \geq 1$ as small as possible. The original paging problem studied by Sleator and Tarjan is a special case of the k-server problem [7]. There is a fairly large literature on competitive algorithms (e.g., [1, 2, 8]). The class of problems most closely related to ours is the online interval packing problem for a single server, where a schedule is a subset of non-overlapping intervals. Lipton and Tomkins [6] study a variant where the input intervals are sorted by their left endpoints. They give a randomized scheduler that is 2-competitive. As we will see, our problem is different from theirs in several ways. Woeginger [10] studied a problem that has several of the features of our problem. Other online interval packing problems can be found in [11, 5].

Outline. In the next section, we formulate our on-line scheduling problem and give two schedulers, `FirstFit` and `EndFit`. `FirstFit` is based on a heuristic which always serves the most important residual request, whereas `EndFit` is based on another heuristic which always serves according to the optimal schedule of the residual requests. We then generalize both schedulers to a class of `Greedy` schedulers in Section 3. The 2-competitiveness of `FirstFit` is shown in Section 4. A considerably harder result is the 2-competitiveness of `EndFit`, shown in Sections 5. In Section 6, we show that all `Greedy` schedulers are 3-competitive. A lower bound of 1.17 on the competitive ratio of any deterministic scheduler is shown in Section 7. In section 8, we give a variant of the scheduling problem under a multi-tasking environment, and discuss its relationship with the original scheduling problem.

2 Problem Formulation

We formalize our problem as an on-line scheduling problem. Each *request* q has four parameters

$$q = (s, t, v, w),$$

where s the *start time*, t the *termination time* (or deadline), v is the *volume* (or size), and w is the *weight*. We require

$$v \geq 0 \quad \text{and} \quad w \geq 0.$$

Write $\mathtt{st}(q)$, $\mathtt{dl}(q)$, $\mathtt{sz}(q)$, $\mathtt{wt}(q)$ for the above parameters of q, respectively. Call the half-open interval $(s, t]$ the *span* of q, written *span*(q). A request q can only be served within its span $(s, t]$, and at any time moment t_0, at most one request can be served.

An *instance* I is a sequence $(q_1, q_2 \ldots q_n)$ of requests where the start times of the q_i's are in increasing order: note that we allow $\mathtt{st}(q_i) = \mathtt{st}(q_{i+1})$ even though we nominally say q_i starts before q_{i+1}. How requests are served is described by the schedule. Formally, A *schedule* for I is a piece-wise constant function

$$H : \mathbb{R} \longrightarrow \{q_1, q_2, \ldots, q_n\} \cup \{\emptyset\}$$

where $|H^{-1}(q_k)| \leq \mathtt{sz}(q_k)$ and $H^{-1}(q_k) \subseteq \mathit{span}(q_k)$ for $k = 1, \ldots, n$. Intuitively, $H(t_0) = q_k$ means the kth request is served at time t_0 and $H(t_0) = \emptyset$ means no request is being served. A time moment t_0 is called a *breakpoint* if H is discontinuous at t_0. (More precisely, for every $\varepsilon > 0$, there exists δ_i $(0 < \delta_i < \varepsilon,\ i = 1, 2)$ such that $H(t_0) = H(t_0 - \delta_1) \neq H(t_0 + \delta_2)$). Since H is piece-wise constant, it has only finitely many breakpoints. A half-open interval of the form $(t_0, t_1]$ is called a *time-slot*. Without loss of generality, we may assume $H^{-1}(q)$ is a finite union of time slots. The merit *merit*(H) of a schedule is

$$\sum_{j=1}^{n} \mathtt{wt}(q_j)|H^{-1}(q_j)|.$$

Relative to a schedule H at any time t_0, we call

$$v' := \mathtt{sz}(q) - |H^{-1}(q) \cap (-\infty, t_0]|$$

the *residual size* of q. A request q is completely served if $v' = 0$. If $v' > 0$ and $\mathtt{st}(q) \leq t_0 \leq \mathtt{dl}(q)$, then we say q is *pending*. The *residue* of a pending q at time t_0 is the modified request $q' = (t_0, \mathtt{dl}(q), v', \mathtt{wt}(q))$.

So each completely served request, the scheduler gains $\mathtt{sz}(q)\mathtt{wt}(q)$ merit points (so weights are multiplicative). Moreover, partially served requests gains a proportional fraction of this merit. This is unlike usual scheduling problems in which partially served requests receive no merit. Our model is meaningful for the "foveated visualization" studied in [4, 3] because a scheduler can reduce the amount of requested visualization data along a finely graduated scale. This can be achieved by reducing two parameters, the *foveal radius* and/or *foveal resolution*. The foveal radius measures how fast the resolution falls away from the foveal center, and the foveal resolution measures the maximum resolution (which is at the foveal center).

Optimal Schedules. We say H is *optimal* for I if $merit(H)$ is maximum among all schedules for I. The existence of optimal schedules is not immediate.

Let I be a sequence of n requests. The $2n$ endpoints of intervals in I are called *natural breakpoints* of I. The time-slot $(s, t]$ defined by two distinct consecutive natural breakpoints s and t is called a *natural slot* of I. There are at most $2n - 1$ natural slots. In the following, let $slot(i)$ denote the i-th natural slot where $slot(1)$ is the earliest natural slot.

Lemma 1 *For every schedule H for I, there is another schedule H' for I with at most $2n^2 + n$ breakpoints such that:*

- merit(H) = merit(H')*;*
- *Within each natural slot, if a request is serviced, it is served in a single time interval.*

Proof. We transform H to H' by modifying its service within each natural slot of I. Within a natural slot, H' can always serve each request in a single (continuous) time interval. It follows that there are at most n breakpoints within a natural slot. Some of the natural breakpoints may now become breakpoints of H'. Since there are at most $2n$ natural breakpoints and at most $2n - 1$ natural slots, H' has at most $2n+(2n-1)n = 2n^2+n$ breakpoints. Finally, note that the merit of H is not changed by this transformation. **Q.E.D.**

Theorem 2 *For all sequences I of n requests, there exists an optimal schedule H^* with at most $2n^2 + n$ breakpoints.*

Proof. Let $\mu^* = \sup_H\{merit(H)\}$ where H range over all schedules for H. Then there is an infinite sequence $\{H_k : k \geq 0\}$ of schedules for I such that $merit(H_k)$ is non-decreasing in k and achieves μ^* in the limit. We now construct a schedule H^* with $merit(H^*) = \mu^*$. For any schedule H for I, define the function

$$\tau_H : \{1, \ldots, 2n - 1\} \times \{1, \ldots, n\} \to \mathbb{R}$$

where $\tau_H(i, j) = |slot(i) \cap H^{-1}(q_j)|$ where q_j is the j-th request in I. Also, write τ_k instead of τ_{H_k}. By going to a subsequence if necessary, we can assume that for all i, j, $\tau_k(i, j)$ is monotone (increasing or decreasing) as $k \to \infty$; we let $\tau^*(i, j)$ denote the limiting value. Choose H'_k to be the schedule obtained from H_k using the

transformation of the preceding lemma. Write τ'_k instead of $\tau_{H'_k}$. Finally, define H^* so that H^* has the form given by the previous lemma, and $\tau_{H^*}(i,j) = \tau^*(i,j)$ for all i,j. Clearly $merit(H^*) = \mu^*$. It remains to show that H^* is a schedule for I. Fix an arbitrary q_j in I. It suffices to check that $|(H^*)^{-1}(q_j)| \leq \mathtt{sz}(q_j)$. For all k, we have $|(H'_k)^{-1}(q_j)| = \sum_{i=1}^{2n-1} \tau'_k(i,j) \leq \mathtt{sz}(q_j)$. Hence $|(H^*)^{-1}(q_j)| = \sum_{i=1}^{2n-1} \tau^*(i,j) \leq \mathtt{sz}(q_j)$.
Q.E.D.

Preemption. It is implicit in the above definitions that the servicing of any request can be preempted as often as we like with no penalty. Hence we may imagine the scheduler to "plan" a schedule based on all the currently residual requests. It services the requests according to this plan until the arrival of a new request. Then it suspends the current plan, recomputes a new plan based on the new set of residual requests, and repeats the process.

Ordering of Requests. The schedulers in this paper make decisions by giving priority to heavier weighted requests. In particular, each arrival of a new request may result in preemption. In case $wt(p) = wt(q)$, we resolve the tie by treating p as "heavier" than q if and only if p starts before q.

Canonical Schedule. A schedule H is *canonical* if, besides satisfying the conditions of Lemma 1, the following hold:

P1. *Serve lighter requests before heavier ones if possible.* If $t_0 < t_1$ and $H(t_0)$ is heavier than $H(t_1)$, then $dl(H(t_0)) < t_1$ or $\mathtt{st}(H(t_1)) > t_0$.

P2. *Local Optimality.* For all t_0 and q_1, if $q_0 = H(t_0)$ is lighter than q_1, and $t_0 \in span(q_1)$, then $|H^{-1}(q_1)| = sz(q)$.

In the above definition, if $H(t) = \emptyset$, we interpret $\emptyset$ to be the request $(-\infty, +\infty, \infty, 0)$. By definition, $\emptyset$ is lightest among all requests.

Lemma 3 *Every schedule H can be transformed into a canonical schedule H' where* merit$(H) \leq$ merit(H').

Proof. First, we can modify H so that it satisfies Lemma 1 and there are no violations of P2. After this, we keep performing "swaps" to remove violation of P1. Our swaps can be specified by two slots $T_0 = (s_0, t_0]$ and slot $T_1 = (s_1, t_1]$ where $s_0 < t_0 \leq s_1 < t_1$. H is constant each T_i, and they represent a violation of P2. We modify H by serving $H(t_1)$ in T_0 and vice-versa. Such a swap does not create any new violations of Lemma 1 after suitable rearrangement within each natural slot. There is also no new violation of P2, and the merit does not change. We choose each swap so that t_1 is maximized, and subject to this, we also maximize $t_1 - s_1$. So in successive swaps, t_1 will strictly decrease. If this process stops after finitely many steps, then H' can be chosen to be the final schedule. Otherwise, by a limiting type argument similar to the proof of Lemma 1, we can construct a canonical schedule H' from this infinite sequence of schedules. **Q.E.D.**

In particular, if H is optimal, H' will be a canonical optimal schedule. Let $\mathtt{opt}(I)$ denote any canonical optimal schedule for I. We do not claim that $\mathtt{opt}(I)$ is unique. Indeed, this is false if weights are not unique. E.g., if $I = (q_0, q_1, q_2)$ where $q_0 = (0, 2, 2, 2)$, $q_1 = (0, 1, 1, 1)$ and $q_2 = (1, 2, 1, 1)$, then there are clearly two canonical optimal schedules. But this is not a counter example under our tie-breaking rules. Below, we prove a case where canonical optimal schedules are unique.

Example 1. The online scheduler that always serve the heaviest residual request at each moment is called the `FirstFit` scheduler. Figure 1 shows the schedule produced by `FirstFit` on an instance of two requests q_1 and q_2. Although this example may appear contrived, we can modify q_2 to $\widetilde{q}_2$ where $\widetilde{q}_2 = (0, 2, 1, 1 + \epsilon)$. As ϵ tends to zero, the `FirstFit` schedule will be the one shown in Figure 1.

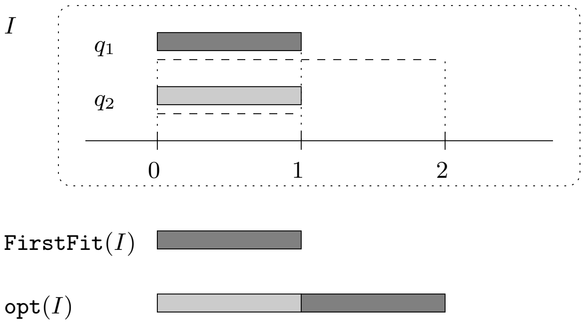

Figure 1: The top figure illustrates the instance I of two requests: $q_1 = (0, 2, 1, 1)$ and $q_2 = (0, 1, 1, 1)$. Each horizontal "dashed" line represents the span of the request. Although $\mathtt{wt}(q_1) = \mathtt{wt}(q_2)$, q_1 is "heavier" than q_2 by our tie-breaking rule. In $\mathtt{opt}(I)$, q_1 and q_2 are served in the time-slots $(1, 2]$ and $(0, 1]$, respectively. However, in $\mathtt{FirstFit}(I)$, only q_1 is served.

The Off-line `EndFit` Scheduler. Consider an online scheduler which always serves according to the optimal schedule for the current set of residual requests. This was first suggested by Estie Arkin[1] who also showed that such an algorithm is 2-competitive. To implement such a scheduler, we can invoke a general off-line algorithm for computing optimal schedules upon each new arrival of a request. But it turns out that a very simple scheduler can be used. This scheduler, on an arbitrary instance I, operates as follows:

[1]Private communication (1997).

> Starting from the heaviest request down to the lightest, allocates each request $q \in I$ to the latest available time-slot(s) within *span*(q).

Call this the `OffEndFit` scheduler. It is an off-line algorithm because it must see the entire set of requests to make its decisions. The next lemma shows that this scheduler is optimal for a special class of inputs.

Lemma 4 *If I is an instance in which all requests have a common starting time, then* `OffEndFit`(I) *is an optimal schedule for I that is both canonical and unique.*

Proof. Let $H := \mathtt{OffEndFit}(I)$ and $H^* := \mathtt{opt}(I)$ be any canonical optimal schedule. First, observe that H has canonical form and, of course, H is uniquely determined by I. Hence it remains to show that H^* is equal to H. By way of contradiction, let $(s_1, t_1]$ be a time slot such that H and H^* are constant on this slot but $q = H(t_1) \neq H^*(t_1) = q^*$. We may assume that t_1 is maximized, so that for all $t > t_1$, $H(t) = H^*(t)$. By symmetry, we may assume that q is heavier than q^*. By property P2 for H^*, we infer that that q was served at some time before t_1. But this contradicts property P1. Hence $H = H^*$. **Q.E.D.**

Example 2. Let `EndFit` be the online scheduler which always serves according to the `OffEndFit` schedule for the residual requests. More precisely, on arrival of a new request, `EndFit` preempts the current service. It computes a new schedule P for the current residual requests using `OffEndFit`, and continues by servicing P. Since all residual requests have a common starting time, P is the canonical optimal schedule. Figure 2 shows the `EndFit` schedule for an instance $I = (q_1, q_2)$. The `EndFit` schedule for this example may appear contrived. To see that it is "correct in the limit", let $I_\epsilon = (q_0, q_1, q_2)$ where $q_0 = (0, 1, 1, \epsilon)$. For any $0 < \epsilon < 1$, the off-line optimal schedule for the current residual requests is unique. As $\epsilon \to 0$, `EndFit`(I_ϵ) approaches the one schedule shown in Figure 2.

3 A Class of Greedy Schedulers

Looking at the behavior of `EndFit` and `FirstFit` on specific examples, it appears that they are complementary in the sense that if `EndFit` performs poorly on an instance, then `FirstFit` will perform well, and vice-versa. This suggests studying some combination of these two heuristics and motivates the generalization to a class of `Greedy` schedulers.

A scheduler S in this class behaves as follows.

(A) At the moment a new request q starts, it suspends the current service (that is, it preempts the currently served request).

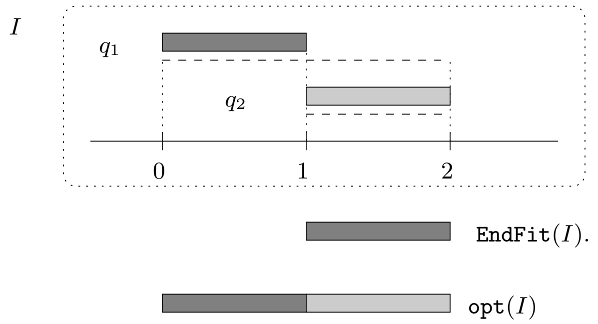

Figure 2: The top figure illustrates the instance I of two requests: $q_1 = (0, 2, 1, 1)$ and $q_2 = (1, 2, 1, 1)$. In the $\texttt{opt}(I)$, q_1 and q_2 are served. However, in $\texttt{EndFit}(I)$, only q_1 is served.

(B) Scheduler S computes a new *plan* H, which is a schedule for the set of residues of currently pending requests. We call H a 'plan' because the scheduler may not carry out the schedule as planned due to the subsequent new requests. The plan H is computed by considering the residues one by one, starting from the heaviest request down to the lightest request. Let p be the request being considered and call $|H^{-1}(p)|$ the allocation to p. The allocation to p is subjected to the following restriction:

($*$) The allocation to p must be maximized. For example, if it is possible to completely allocate p, the whole of p must be allocated. However, there is no restriction on where p is allocated. Time-slots, once allocated, are not subsequently revised in creating this plan.

(C) It carries out the plan until a new request starts, whereupon we go back to step (A).

Let the plan computed after step (B) be $\mathit{Plan}^+(S, I, q)$ and $\mathit{Plan}^-(S, I, q)$ be the original plan just before step (B) is executed. Thus $\mathit{Plan}^-(S, I, q)$ is actually $\mathit{Plan}^+(S, I, q')$, where q' is the request which starts just before q.

Different members of the `Greedy` class differ only in their strategies for (B) subjected to the restriction($*$). Note that our first example `FirstFit` is a `Greedy` scheduler: the request p in ($*$) is allocated in the earliest possible time-slots.

The second example `EndFit` is also a greedy scheduler. Its strategy for (B) is rather counter-intuitive: the request p is allocated in the *latest* possible time slots.

It is helpful to visualize the relationship between $\mathit{Plan}^-(\texttt{EndFit}, I, p)$ and $\mathit{Plan}^+(\texttt{EndFit}, I, p)$. To obtain $\mathit{Plan}^+(\texttt{EndFit}, I, p)$ from $\mathit{Plan}^-(\texttt{EndFit}, I, p)$, the scheduler first finds the location to "insert" the newly started request p; it then "squeezes" p in by "pushing" the original allocated requests leftward (to earlier time-slots). In so doing, some requests may be "pushed out" from the plan. The plans produced by `FirstFit` can be visualized in the similar way except that the "push" is in the opposite direction.

4 Competitive Ratio of `FirstFit`

Example 1 (Figure 1) shows that the competitive ratio of `FirstFit` is at least 2. In fact, we will show that `FirstFit` is 2-competitive. Before presenting the proof, let us give two definitions.

Charging Scheme. Let H, H_1 and H_2 be schedules for an instance I. We often need to argue that the merit of H is no larger than the sum of the merits of H_1 and H_2. Our approach is to *charge* a portion of H to H_1 and the remaining to H_2. Intuitively, the charging process can be viewed as first cutting H_1 and H_2 into pieces and then piecing them together again to form another piecewise-constant function $H_{\mathtt{chg}}$. Each piece, after cutting, may be translated before being placed in their slot in $H_{\mathtt{chg}}$. As it may turn out that $|H^{-1}_{\mathtt{chg}}(q)| > \mathtt{sz}(q)$ for some request q, $H_{\mathtt{chg}}$ is not necessarily a schedule. The cut-and-paste is done in a way that for all t, $\mathtt{wt}(H_{\mathtt{chg}}(t)) \geq \mathtt{wt}(H(t))$. Therefore,

$$merit(H) \leq merit(H_{\mathtt{chg}}) \leq merit(H_1) + merit(H_2). \tag{1}$$

In particular, if $H_1 = H_2$, then we have

$$merit(H) \leq 2 \cdot merit(H_1).$$

When we use the phrase: "charge $(s', e']$ from H to H_1 at $(s, e]$", we mean that a piece $(s, e]$ is cut from H_1 and placed into the slot $(s', e']$ in $H_{\mathtt{chg}}$. Equivalently, if $H^{-1}(q)$ is the interval $(s', e']$, we may say that the request q is charged from H to H_1 at $(s, e]$ (see Figure 3). Thus, to show (1), we need to charge each time slot of H to either H_1 or H_2, and ensure that no part of H_1 or H_2 is charged more than one.

Intactness. A request q is *intact* in a schedule H if $H^{-1}(q)$ is connected, and either $|H^{-1}(q)| = 0$ or $\mathtt{sz}(q)$. An instance I is intact in a schedule H if each request q in I is intact. Most of our proofs will be simplified if we assume intactness. Fortunately, since our scheduling problem is preemptive, we could break a request into pieces so as to achieve intactness. In most of our proofs, we assume that the instance is intact in the schedules in question.

Theorem 5 *For any instance I,*

$$\mathrm{merit}(\mathtt{opt}(I)) \leq 2 \cdot \mathrm{merit}(\mathtt{FirstFit}(I)).$$

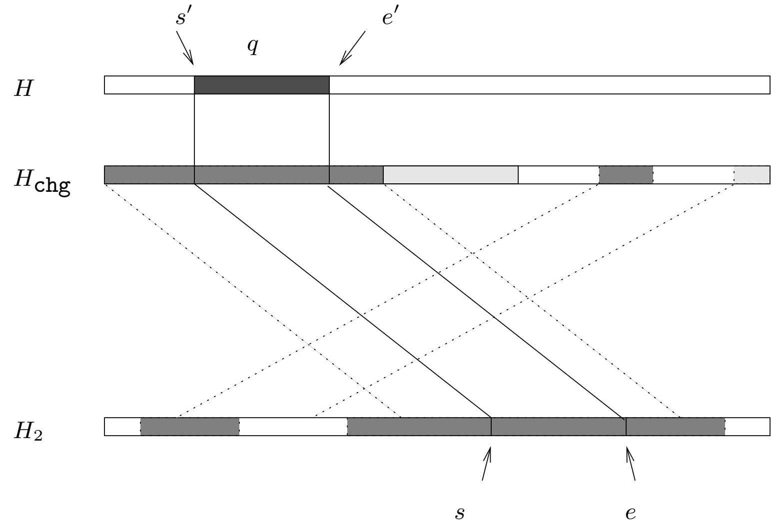

Figure 3: Charging the request q in H to H_1 at $(s, e]$.

Proof.

Given an instance I, let $H_{\texttt{ff}} := \texttt{FirstFit}(I)$ and $H_{\texttt{opt}} := \texttt{opt}(I)$. We can assume that I is intact in both $H_{\texttt{ff}}$ and $H_{\texttt{opt}}$.

Let H_0 be an identical copy of $H_{\texttt{ff}}$. We want to charge requests served in $H_{\texttt{opt}}$ to H_0 and $H_{\texttt{ff}}$. Let $\{t_1, t_2, \ldots, t_m\}$ be the distinct breakpoints in $H_{\texttt{opt}}$, where $t_i < t_j$ if and only if $i < j$.

For each t_i, starting from $i := 1$ to $m - 1$, consider the time-slot $(t_i, t_{i+1}]$. Let $q_{\texttt{opt}} := H_{\texttt{opt}}(t_{i+1})$. Let $q_{\texttt{ff}}$ be the lightest request served during $(t_i, t_{t+1}]$ by `FirstFit`. There are two cases:

1. If the request $q_{\texttt{ff}}$ is not lighter than $q_{\texttt{opt}}$, charge $q_{\texttt{opt}}$ from $H_{\texttt{opt}}$ to $H_{\texttt{ff}}$ at $(t_i, t_{i+1}]$.

2. Otherwise, charge $q_{\texttt{opt}}$ from $H_{\texttt{opt}}$ to H_0 at $H_0^{-1}(q_{\texttt{opt}})$.

We have to show that in the second case, $|H_0^{-1}(q_{\texttt{opt}})| \geq |(t_i, t_{i+1}]|$. In the first place, why is the weight of $q_{\texttt{ff}}$ lighter? The request $q_{\texttt{ff}}$ is chosen by `FirstFit` because it is the heaviest request among the pending requests. This implies that $q_{\texttt{opt}}$ is not a pending request, even though t_i is in the span of $q_{\texttt{opt}}$. So $q_{\texttt{opt}}$ must have been completely served by `FirstFit`. This implies that $|H_0^{-1}(q_{\texttt{opt}})| \geq |(t_i, t_{i+1}]|$.

Q.E.D.

5 Competitive Ratio of `EndFit`

Example 2 (Figure 2) shows that the competitive ratio of `EndFit` is at least 2. We now show that this constant is the best possible.

The upper bound proof is considerably more subtle than the proof for `FirstFit` in the last section. The key result is Theorem 11 below which formalizes this observation about `EndFit`: it never hurts the performance of `EndFit` to have a request started at an earlier time. For example, in Figure 2, the performance of `EndFit` will improve if the request q_2 starts at an earlier time. The analogous lemma fails for `FirstFit`. For example, in Figure 1, the performance of `FirstFit` would improve if q_1 starts at a time later than 0.

5.1 Trimmed instances and level sets

A request $\widetilde{q}$ is a *trimmed* version of q if $\mathtt{st}(\widetilde{q}) \geq \mathtt{st}(q)$, $\mathtt{dl}(\widetilde{q}) = \mathtt{dl}(q)$, $\mathtt{wt}(\widetilde{q}) = \mathtt{wt}(\widetilde{q})$ and $\mathtt{sz}(\widetilde{q}) \leq \mathtt{sz}(q)$. Thus, a trimmed version of q may start later than the original q. The main motivation for this definition comes from the fact that the residual request of any request q is a trimmed version of q.

An instance $\widetilde{I}$ is a trimmed instance of I if there is a one-one (not necessarily onto) mapping from $\widetilde{I}$ to I such that any $\widetilde{q}$ in $\widetilde{I}$ is a trimmed version of its corresponding request in I. Clearly, $merit(\mathtt{opt}(\widetilde{I})) \leq merit(\mathtt{opt}(I))$. We want to show that a similar relationship also holds for $\mathtt{EndFit}(\widetilde{I})$ and $\mathtt{EndFit}(I)$.

To analyze EndFit, we introduce the notion of "level sets". Let H be a schedule for I and q be any request. We do not require q to belong to I but it must be totally ordered with respect to the requests in I. Define the *q-level set* of H to be

$$\mathtt{Level}_H(q) := \{t \in \mathbb{R} : \mathtt{wt}(H(t)) \geq \mathtt{wt}(q)\}.$$

Also define the related sets,

$$\mathtt{Level}^+_H(q) := \{t \in \mathbb{R} : \mathtt{wt}(H(t)) > \mathtt{wt}(q)\}$$

and

$$\mathtt{Level}^0_H(q) := \{t \in \mathbb{R} : \mathtt{wt}(H(t)) = \mathtt{wt}(q)\}.$$

In these three notions of level sets, the comparison of weights are not really comparison of real numbers, but uses our tie-breaking conventions for weights of requests. E.g. "$\mathtt{wt}(H(t)) > \mathtt{wt}(q)$" should really read "request $H(t)$ is heavier than request q". For this reason, we do not define "$\mathtt{Level}_H(w)$" where w is a real number because the real number w does not carry any tie-breaking information.

Clearly, $\mathtt{Level}_H(q) = \mathtt{Level}^+_F(q) \cup \mathtt{Level}^0_F(q)$. In case H is equal to $\mathtt{OffEndFit}(I)$, we will write $\mathtt{Level}_I(q), \mathtt{Level}^+_I(q), \mathtt{Level}^0_I(q)$ instead of $\mathtt{Level}_H(q), \mathtt{Level}^+_H(q), \mathtt{Level}^0_H(q)$, respectively. It is important that note that we use `OffEndFit`, not `EndFit` in this definition.

We break up the operations of `OffEndFit` using the following device: by a *level set* L we mean a finite union of time slots. If q is a request, define $\mathtt{EF}(q, L) \subseteq \mathbb{R}$ to be the time slots that are scheduled for q by the `OffEndFit` scheduler, assuming that L have already been scheduled. Thus $\mathtt{EF}(q, L) \cap L$ is empty.

If $q \in I$, then

$$\texttt{Level}_I^0(q) = \texttt{EF}(q, \texttt{Level}_I^+(q)).$$

Hence,

$$\texttt{Level}_I(q) = \texttt{EF}(q, \texttt{Level}_I^+(q)) \cup \texttt{Level}_I^+(q).$$

Writing $s = \inf \texttt{Level}_I^0(q)$, we can further express this as

$$\texttt{Level}_I(q) = (s, \texttt{dl}(q)] \cup \texttt{Level}_I^+(q).$$

In general, for any level set L and any q, if $s = \inf \texttt{EF}(q, L)$ then

$$\texttt{EF}(q, L) \cup L = (s, \texttt{dl}(q)] \cup L. \tag{2}$$

Lemma 6 *Let $L' \subseteq L$ be level sets. If $\widetilde{q}$ is a trimmed version of q then*

$$\texttt{EF}(\widetilde{q}, L') \cup L' \subseteq \texttt{EF}(q, L) \cup L.$$

Proof. Let $C = \mathit{span}(q) \setminus L$ and $C' = \mathit{span}(\widetilde{q}) \setminus L'$. For any $s \in \mathbb{R}$, let $C_s = \{t \in C : t > s\}$. Similarly define $C'_s \subseteq C'$. Because $\widetilde{q}$ is a trimmed version of q, for all $s \geq \texttt{st}(\widetilde{q})$ (which is $\geq \texttt{st}(q)$), we have

$$C_s \subseteq C'_s.$$

But $\texttt{EF}(\widetilde{q}, L') = C'_{s_0}$ for some $s_0 \geq \texttt{st}(\widetilde{q})$, and similarly $\texttt{EF}(q, L) = C_{s_1}$ for some $s_1 \geq \texttt{st}(q)$. Here, we pick the largest possible value for s_0 and s_1. According to (2)

$$\texttt{EF}(\widetilde{q}, L') \cup L' = (s_0, \texttt{dl}(\widetilde{q})] \cup L', \qquad \texttt{EF}(q, L) \cup L = (s_1, \texttt{dl}(q)] \cup L. \tag{3}$$

Hence the lemma would follow if $(s_0, \texttt{dl}(\widetilde{q})] \subseteq (s_1, \texttt{dl}(q)]$. Since $\texttt{dl}(q) = \texttt{dl}(\widetilde{q})$, it is enough to show

$$s_1 \leq s_0.$$

To see this, note that

$$|C_{s_0}| \leq |C'_{s_0}| \leq \texttt{sz}(\widetilde{q}) \leq \texttt{sz}(q)$$

and

$$s_0 \geq \texttt{st}(\widetilde{q}) \geq \texttt{st}(q)$$

using the fact that $\widetilde{q}$ is a trimmed version of q. If $|C_{s_0}| = \texttt{sz}(q)$ or $s_0 = \texttt{st}(q)$ then we must have $s_1 = s_0$ Otherwise, suppose $|C_{s_0}| < \texttt{sz}(q)$ and $s_0 > \texttt{st}(q)$. Then it is immediate that $s_1 < s_0$. **Q.E.D.**

5.2 Domination

In the following, let $\widetilde{I}$ be a trimmed version of I. Our goal is to prove that

$$merit(\texttt{EndFit}(\widetilde{I})) \leq merit(\texttt{EndFit}(I))$$

where $\widetilde{I}$ is a trimmed version of I. The key idea is to study the domination relation: for two schedules, we say that H *dominates* H' if for all t, $\texttt{wt}(H(t)) \geq \texttt{wt}(H'(t))$. For two input instances, we say I *dominates* I' if $\texttt{OffEndFit}(I)$ dominates $\texttt{OffEndFit}(I')$. Again, note that domination between instances is defined in terms of $\texttt{OffEndFit}$, not $\texttt{EndFit}$. Domination can be reduced to inclusion relationship between level sets: thus, I dominates I' if and only if for all $q' \in I'$,

$$\texttt{Level}_{I'}(q') \subseteq \texttt{Level}_I(q').$$

Lemma 7 *If $\widetilde{I}$ is a trimmed version of I then I dominates $\widetilde{I}$.*

Proof. We use induction on the number of requests in I. Suppose q is the lightest request in I and $\widetilde{q} \in \widetilde{I}$ is the corresponding trimmed version of q. Just before $\texttt{OffEndFit}$ schedules q, we have a partial schedule H for all the requests in $I \setminus \{q\}$. Let H' be similarly defined with respect to $\widetilde{I}$ and $\widetilde{q}$. Let $L := \texttt{Level}^+_I(q) = \texttt{Level}^+_H(q)$ and $L' := \texttt{Level}^+_{\widetilde{I}}(\widetilde{q}) = \texttt{Level}^+_{H'}(\widetilde{q})$. By induction, H dominates H'. To show that I dominates $\widetilde{I}$, it remains to show

$$\texttt{Level}_{\widetilde{I}}(\widetilde{q}) \subseteq \texttt{Level}_I(q). \tag{4}$$

But $\texttt{Level}_{\widetilde{I}}(\widetilde{q}) = \texttt{EF}(\widetilde{q}, L') \cup L'$ and $\texttt{Level}_I(w) = \texttt{EF}(q, L) \cup L$. Since, $L' \subseteq L$ and $\widetilde{q}$ is a trimmed version of q, Lemma 6 implies (4). **Q.E.D.**

As corollary, if $\widetilde{I}$ is obtained by deleting a request from I, then I dominates $\widetilde{I}$. Note that the preceding lemma does not depend on the requests in I or $\widetilde{I}$ having a common start time.

Next, for any time $t \in \mathbb{R}$, let $I|_t$ denote the *residual instance* of I when the schedule $\texttt{EndFit}(I)$ has been serviced up to time t. More precisely, each $q \in I$ with $\texttt{dl}(q) \leq t$ is deleted. Otherwise, q is replaced by q_t which is the residue of q at time t in $\texttt{EndFit}(I)$. If q has been completely served by time t, then its residue has zero size, $\texttt{sz}(q_t) = 0$. Such requests are called *null requests*. For technical reasons, we do not discard null requests from $I|_t$. Clearly all requests in $I|_t$ starts at or after time t.

To motivate the next definition, observe that if I dominates $\widetilde{I}$, then for all t, $I|_t$ dominates $\widetilde{I}|_t$. Unfortunately, $\widetilde{I}|_t$ may no longer be a trimmed version of $I|_t$, even if $\widetilde{I}$ is a trimmed version of I. For example, let $I = (q_0, q_1) = ((0,3,2,1),(1,2,1,2))$ and $\widetilde{I} = (q'_0, q_1) = ((1,3,2,1),(1,2,1,2))$. If $t = 1$, $I|_t = (q_2, q_1) = ((1,3,1,1),(1,2,1,2))$ while $\widetilde{I}|_t = (q'_0, q_1) = ((1,3,2,1),(1,2,1,2))$. Then $\widetilde{I}|_t$ is not a trimmed version of $I|_t$ because q'_0 is not a trimmed version of q_2. According to the definitions below, it turns out that q_2 "spans" q'_0, and q_2 is "saturated" in $I|_t$.

A request $q \in I$ is *saturated* in I if

$$|span(q) \setminus \mathtt{Level}_I^+(q)| \leq \mathtt{sz}(q).$$

Note that $span(q) \setminus \mathtt{Level}_I^+(q)$ comprise all the times available for scheduling q relative to I. A request q *spans* another request q' if $span(q') \subseteq span(q)$. For example, if q' is a trimmed version of q then q spans q'. We define a binary relation "$\succeq$" on instances as follows: if I and J are instances, we write

$$I \succeq J$$

if $I = (q_1, \ldots, q_n)$ and $J = (q'_1, \ldots, q'_n)$ and for all $i = 1, \ldots, n$, $\mathtt{wt}(q_i) = \mathtt{wt}(q'_i)$ and

(a) either q'_i is a trimmed version of q_i, or

(b) q_i spans q'_i and q_i is saturated in I.

If $f : I \to J$ such that $f(q_i) = q'_i$ for all i, then we say "$I \succeq J$ *via* f". Note that if J is a trimmed version of I and $|I| = |J|$, then $I \succeq J$. The following is immediate from the definition of $\succeq$:

Lemma 8 *Let $f : I \to J$ be a bijection and $f(q) = \widetilde{q}$ for some $q \in I$. Let $I' = I \setminus \{q\}$ and $J' = J \setminus \{\widetilde{q}\}$ and $f' : I' \to J'$ be the restriction of f to I'. If $I' \succeq J'$ via f' then $I \succeq J$ via f.*

The next lemma shows that the relation $\succeq$ is preserved by residual instances:

Lemma 9 *If $I \succeq J$ then for all t, $I|_t \succeq J|_t$.*

Proof. Suppose $I \succeq J$ via f. Let $q \in I$ and $\widetilde{q} = f(q)$. There are two cases: (a) Suppose $\widetilde{q}$ is a trimmed version of q. Let $\widetilde{q}_t$ and q_t be the corresponding requests in $J|_t$ and in $I|_t$. Clearly, q_t spans $\widetilde{q}_t$. If $\mathtt{sz}(q_t) \geq \mathtt{sz}(\widetilde{q}_t)$ then $\widetilde{q}_t$ is a trimmed version of q_t. Otherwise, it must be the case that q_t became saturated in $I|_t$. (b) Suppose q spans $\widetilde{q}$ and q is saturated in I. Then it is easy to see that q_t spans $\widetilde{q}_t$ and q_t is saturated in $I|_t$. **Q.E.D.**

The next result connects the relation $\succeq$ to domination:

Lemma 10 *If $I \succeq J$ then I dominates J.*

Proof. Let $I \succeq J$ via $f : I \to J$. We use induction on the number $|I| = |J|$ of requests in I or J. Suppose $|I| = 1$. In case condition (a) holds in the definition of $I \succeq J$, then clearly I dominates J. In case condition (b) holds, then q is saturated in I implies $\mathtt{dl}(q) - \mathtt{st}(q) = \mathtt{sz}(q)$. Combined with the fact that q spans $f(q)$, we conclude that $\mathtt{sz}(f(q)) \leq \mathtt{sz}(q)$. So again, I dominates J. Next assume $|I| > 1$ and let q be the lightest request in I. So $f(q)$ is the lightest in J. Let $J' = J \setminus \{f(q)\}$

and $I' = I \setminus \{q\}$. It remains to prove that $\mathtt{Level}_J(f(q)) \subseteq \mathtt{Level}_I(q)$. By induction, I' dominates J'. We have

$$\mathtt{Level}^+_J(f(q)) = \mathtt{Level}^+_{J'}(f(q)) \subseteq \mathtt{Level}^+_{I'}(q) = \mathtt{Level}^+_I(q). \tag{5}$$

If $f(q)$ is a trimmed version of q, then it follows from (5) and Lemma 6 that $\mathtt{Level}_J(f(q)) \subseteq \mathtt{Level}_I(q)$. Otherwise, q spans $f(q)$ and q is saturated in I. This means

$$\begin{aligned}
\mathtt{Level}_I(q) &= \mathtt{Level}^+_I(q) \cup \mathit{span}(q) \qquad \text{(since } q \text{ is saturated in } I\text{)} && (6)\\
&\supseteq \mathtt{Level}^+_J(f(q)) \cup \mathit{span}(f(q)) \qquad \text{(since } q \text{ spans } f(q)\text{) and by (5)} && (7)\\
&\supseteq \mathtt{Level}_J(f(q)). && (8)
\end{aligned}$$

Q.E.D.

We come to the main result.

Theorem 11 *If $\widetilde{I}$ is a trimmed version of I then*

$$\text{merit}(\mathtt{EndFit}(\widetilde{I})) \leq \text{merit}(\mathtt{EndFit}(I)).$$

Proof. It sufficient to prove this result in the case where I and $\widetilde{I}$ are identical except that some $q \in I$ is replaced by a trimmed version $\widetilde{q}$ in $\widetilde{I}$. Let

$$s_0 \leq s_1 \leq \cdots \leq s_{m-1} \leq s_m, \qquad (m \geq 0) \tag{9}$$

be the sequence of start times of all $m+1$ requests in I. There are schedules $H_0, H_1 \ldots H_m$ such that in the time slot $(s_i, s_{i+1}]$, the requests of I are serviced according to H_i (assume $s_{m+1} = \infty$). Each H_i is an `OffEndFit` schedule for a suitable instance I_i. (So H_i is just a "plan".) Let m_i be the merit obtained by servicing H_i in the time slot $(s_i, s_{i+1}]$. Then

$$\mathit{merit}(\mathtt{EndFit}(I)) = \sum_{i=0}^{m} m_i. \tag{10}$$

Now, there is a corresponding sequence of start times

$$t_0 \leq t_1 \leq \cdots \leq t_{m-1} \leq t_m \tag{11}$$

of requests in $\widetilde{I}$. Note that the s_i's and t_j's are basically identical except that $\mathtt{st}(q)$ in the list (9) is replaced by $\mathtt{st}(\widetilde{q})$ in the list (11). We will make the lists (9) and (11) identical,

$$s_i = t_i \quad (i = 0, \ldots, m), \tag{12}$$

simply by adding a null request with start time $\mathtt{st}(\widetilde{q})$ into I, and by adding a null request with start time $\mathtt{st}(q)$ into $\widetilde{I}$. Again, let $\widetilde{H}_i$ be the plan in the EndFit servicing of $\widetilde{I}$ at time s_i; so $\widetilde{H}_i$ is the `OffEndFit` schedule for a suitable instance $\widetilde{I}_i$. Let

$merit(\texttt{EndFit}(\widetilde{I})) = \sum_{j=0}^{m} n_j$ be the equation for $\widetilde{I}$ corresponding to (10). Our theorem follows if we show that $m_i \geq n_i$ for all i.

Let i_0 and i_1 be the indices such that $s_{i_0} = \texttt{st}(q)$ and $s_{i_1} = \texttt{st}(\widetilde{q})$. We may assume that $i_0 < i_1$: this is clear in case $\texttt{st}(q) < \texttt{st}(\widetilde{q})$. Even if $\texttt{st}(q) = \texttt{st}(\widetilde{q})$, we can arbitrarily assume q started "just before" $\widetilde{q}$.

Now each I_i is a residual instance of I_{i-1}, plus a new request q_i that starts at time s_i. That is,

$$I_i = (I_{i-1})|_{s_i} \cup \{q_i\}.$$

We say that q_i is "inserted" into I_i. In case $i = i_1$, the request q_i is a suitable null request. Similarly,

$$\widetilde{I}_i = (\widetilde{I}_{i-1})|_{s_i} \cup \{\widetilde{q}_i\}$$

where $\widetilde{q}_i$ is null request in case $i = i_0$.

Our goal of showing $m_i \geq n_i$ can be reduced to showing I_i dominates $\widetilde{I}_i$. But for the induction to carry through, we need to maintain stronger assertion (via Lemma 10), that for all i,

$$I_i \succeq \widetilde{I}_i. \tag{13}$$

It is not hard to see that (13) holds for all $i < i_1$. In fact, $I_i = \widetilde{I}_i$ for $i < i_0$. More generally, for all i except $i = i_1$, the request $\widetilde{q}_i$ inserted into $\widetilde{I}_i$ is a trimmed version of the request q_i inserted into I_i. By an application of Lemma 8 and Lemma 9, we conclude from $I_i \succeq \widetilde{I}_i$ that $I_{i+1} \succeq \widetilde{I}_{i+1}$.

It remains to show that (13) holds when when $i = i_1$. Write J_1 for $(I_{i_1-1})|_{s_1}$ and $\widetilde{J}_1$ for $(\widetilde{I}_{i_1-1})|_{s_1}$. Since $I_{i_1-1} \succeq \widetilde{J}_{i_1-1}$, we conclude that

$$J_1 \succeq \widetilde{J}_1(\text{via } f_1) \tag{14}$$

where f_1 is naturally defined. In J_{i_1} we have a request r that could be traced back to $q = q_{i_0}$ at time s_{i_0} (so r is really $q|_{s_1}$). Also, $f_1(r)$ is traced back to the null request that we inserted into $\widetilde{I}$ in order to ensure (12). We want to show that $I_{i_1} \succeq \widetilde{I}_{i_1}$ via the natural f which extends f_1. Unfortunately, f maps a null request, q_{i_1}, to $\widetilde{q} = \widetilde{q}_{i_1}$. But we had noted that f_1, and hence f, maps r to a null request $f_1(r)$. We will modify f so that r is mapped to $\widetilde{q}$ and null request q_{i_1} is mapped to $f_1(r)$. This change amounts to transposing r and q_{i_1} in I_{i_1}, and is harmless. We now claim that $I_{i_1} \succeq \widetilde{I}_{i_1}$ via f. In view of (14), we only need to show that r and $f(r) = \widetilde{q}$ are related as required by the $\succeq$ relation. There are two cases: (a) If q has not been served by $\texttt{EndFit}(I)$ up to time s_1 then $\widetilde{q}$ is just a trimmed version of r. (b) If $\texttt{EndFit}(I)$ had already started serving q by time s_1, then r is saturated in I_{i_1}. It is also obvious that r spans $\widetilde{q}$. **Q.E.D.**

5.3 `EndFit` is 2-competitive

We now use Theorem 11 to show that `EndFit` is 2-competitive.

Theorem 12 *For any instance I,*

$$\text{merit}(\texttt{opt}(I)) \leq 2 \cdot \text{merit}(\texttt{EndFit}(I)).$$

Proof. We can assume that I is intact in $\texttt{opt}(I)$. Let $\widetilde{I}$ be the trimmed instance of I such that for any request $q \in I$, if $(\texttt{opt}(I))^{-1}(q) = (t_1, t_2]$, then the corresponding trimmed request $\widetilde{q}$ satisfies $\texttt{st}(\widetilde{q}) := t_1$ and $\texttt{sz}(\widetilde{q}) := t_2 - t_1$; otherwise if $\texttt{opt}(I)^{-1}(q) = \emptyset$, then $\widetilde{q}$ satisfies $\texttt{sz}(\widetilde{q}) := 0$. We can further assume that $\widetilde{I}$ is intact in all the plans of $\texttt{EndFit}$ with $\widetilde{I}$. By definition, we have

$$\textit{merit}(\texttt{opt}(I)) = \textit{merit}(\texttt{opt}(\widetilde{I})). \tag{15}$$

The instance $\widetilde{I}$ has the nice property that requests arrive at a "constant rate", that is, if a request q starts at time t, then no other request starts during $(t, t + \texttt{sz}(q))$. Let H_1 and H_2 be two identical copies of $\texttt{EndFit}(\widetilde{I})$. Our theorem is proved if we show how to charge requests in $\texttt{opt}(\widetilde{I})$ to H_1 and H_2.

Consider a request q in $\widetilde{I}$. Upon arrival of q, there are two cases.

1. If q is allocated in the new plan $\textit{Plan}^+(\texttt{EndFit}, \widetilde{I}, q)$, then it is possible that there are some requests which are originally allocated in $\textit{Plan}^-(\texttt{EndFit}, \widetilde{I}, q)$, but not in the new plan. Call these requests the *ousted requests*.

2. Otherwise, call q the ousted request.

Let s be the total size of the ousted requests. Note that $s \leq \texttt{sz}(q)$ and the total merit of the ousted requests is not more than the total merit of the requests allocated in $(\texttt{st}(q), \texttt{st}(q) + s]$ in $\textit{Plan}^+(\texttt{EndFit}, \widetilde{I}, q)$. Furthermore, the new plan will be carried out without interruption at least until $\texttt{st}(q) + \texttt{sz}(q)$. Charge the ousted requests to H_2 at $(\texttt{st}(q), \texttt{st}(q) + \texttt{sz}(q)]$ and the served requests during $(\texttt{st}(q), \texttt{st}(q) + \texttt{sz}(q)]$ to H_1 at $(\texttt{st}(q), \texttt{st}(q) + \texttt{sz}(q)]$. The above is a valid charging scheme. Thus, we have

$$2 \cdot \textit{merit}(\texttt{EndFit}(\widetilde{I})) \geq \textit{merit}(\texttt{opt}(\widetilde{I})).$$

By Theorem 11 and (15), we have

$$2 \cdot \textit{merit}(\texttt{EndFit}(I)) \geq \textit{merit}(\texttt{opt}(I)).$$

Q.E.D.

6 Competitive Ratio of `Greedy`

Given any `Greedy` scheduler S, it is easy to construct an instance with two requests to show that the competive ratio of S at least 2. Since `EndFit` and `FirstFit` are both 2-competitive, this lower bound for greedy schedulers cannot be improved. By combining the counter examples for `FirstFit` and `EndFit`, we can find a `Greedy` scheduler whose competitive ratio ≥ 3. In this section, we prove that this bound of 3 cannot be improved.

Theorem 13 *Every* `Greedy` *scheduler is 3-competitive.*

Proof.

We will fix an instance I and a greedy scheduler S. Our proof will construct a schedule H such that $merit(H) \leq merit(S(I))$. In fact, we will define another scheduler $\widetilde{S}$ and instance $\widetilde{I}$ such that $H = \widetilde{S}(\widetilde{I})$. The instance $\widetilde{I}$ is choosen in a way such that $merit(\mathtt{opt}(I)) = merit(\mathtt{opt}(\widetilde{I}))$. We exploit the properties of $\widetilde{S}$ and $\widetilde{I}$ to show that

$$3 \cdot merit(\widetilde{S}(\widetilde{I})) \geq merit(\mathtt{opt}(\widetilde{I})).$$

This implies our theorem.

Part I. We may assume that the given instance I is intact in $\mathtt{opt}(I)$, $S(I)$ and all plans $Plan^{+}(S, I, q)$.

Let $Q_{\mathtt{opt}}$ be the set of requests that are served in $\mathtt{opt}(I)$ and let $Q_{\mathtt{gdy}}$ be the set of requests that are served in $S(I)$. Let $\widetilde{I}$ be a trimmed instance of I defined as follow:

1. For each $q \in Q_{\mathtt{gdy}}$, the corresponding trimmed request $\widetilde{q}$ satisfies $\mathtt{st}(\widetilde{q}) := s'$ and $\mathtt{sz}(\widetilde{q}) := \mathtt{sz}(q)$, where $(s', t'] := S(I)^{-1}(q)$;

2. For each $q \in I - (Q_{\mathtt{gdy}} \cup Q_{\mathtt{opt}})$, $\mathtt{sz}(\widetilde{q}) := 0$;

3. For each $q \in Q_{\mathtt{opt}} - Q_{\mathtt{gdy}}$, the corresponding trimmed request $\widetilde{q}$ has start-time $\mathtt{st}(\widetilde{q}) := u'$ and $\mathtt{sz}(\widetilde{q}) := \mathtt{sz}(q)$ where $(u', v'] := \mathtt{opt}(I)^{-1}(q)$.

Let $\widetilde{Q}_{\mathtt{opt}}$ and $\widetilde{Q}_{\mathtt{gdy}} \subseteq \widetilde{I}$ be the corresponding trimmed instance $Q_{\mathtt{opt}}$ and $Q_{\mathtt{gdy}}$ respectively.

Now, we describe how H is constructed. Let $\widetilde{S}$ be a scheduler that always allocates a request $\widetilde{q} \in \widetilde{Q}_{\mathtt{gdy}}$ in the time-slot $[\mathtt{st}(\widetilde{q}), \mathtt{st}(\widetilde{q}) + \mathtt{sz}(\widetilde{q}))$. For the remaining residual requests it allocates them in the way same as `EndFit`. By construction, at any time, there is at most one residual request $\widetilde{q}$ from $\widetilde{Q}_{\mathtt{gdy}}$. This request $\widetilde{q}$ is allocated in the earliest possible time-slot. All the remaining residual requests are from $\widetilde{Q}_{\mathtt{opt}} - \widetilde{Q}_{\mathtt{gdy}}$, and are allocated after time $\mathtt{st}(\widetilde{q}) + \mathtt{sz}(\widetilde{q})$. Since `EndFit` gives the optimal schedule for residual requests, the allocation after $\mathtt{st}(\widetilde{q}) + \mathtt{sz}(\widetilde{q})$ is optimal.

Clearly, $merit(\widetilde{S}(\widetilde{I})) = merit(S(I))$. Note that we need to know $\mathtt{opt}(I)$ prior to the servicing of $\widetilde{S}$. Hence we do not claim, nor need, $\widetilde{S}$ to be an on-line algorithm.

Part II. Now we exploit the restricted form imposed on $\widetilde{S}$. Consider a request $\widetilde{q}$. Due to the arrival of $\widetilde{q}$, there are some requests in $Plan^{-}(\widetilde{S}, \widetilde{I}, \widetilde{q})$ that are not in $Plan^{+}(\widetilde{S}, \widetilde{I}, \widetilde{q})$. We call these requests the *ousted* requests and say that they are being ousted by $\widetilde{q}$.

The total size of these ousted requests is less than or equal to $\mathtt{sz}(\widetilde{q})$. Furthermore, their weights are not larger than the weights of the requests allocated in $(\mathtt{st}(\widetilde{q}), \mathtt{st}(\widetilde{q}) + \mathtt{sz}(\widetilde{q})]$. To see that the weight of an ousted request $\widetilde{p}$ is not larger, we consider the original S. Let $\widetilde{q}_1 \in \widetilde{Q}_{\text{gdy}}$ be the request allocated in $(\mathtt{st}(\widetilde{q}), \mathtt{st}(\widetilde{q}) + \mathtt{sz}(\widetilde{q})]$. Since $\widetilde{p} \notin \widetilde{Q}_{\text{gdy}}$, p is not served by S. The trimmed request $\widetilde{p}$ is being ousted by $\widetilde{S}$ because there are no available time-slot in $(\mathtt{st}(\widetilde{q}_1) + \mathtt{sz}(\widetilde{q}_1), \mathtt{dl}(\widetilde{p})]$. Since $\widetilde{S}$ optimally allocates the residuals into this time-slot and all residuals have common start time, thus it is impossible for S to allocate p in this corresponding time-slot. If $\mathtt{wt}(p) > \mathtt{wt}(q_1)$, then S has to allocate p before time $\mathtt{st}(\widetilde{q}_1) + \mathtt{sz}(\widetilde{q}_1)$. This contradicts the fact that q_1 is served by S. Thus, p is lighter than q_1, which implies $\widetilde{p}$ is lighter than $\widetilde{q}_1$.

Part III. Let H_1, H_2 and H_3 be three copies of $\widetilde{S}(\widetilde{I})$. We want to charge $\mathtt{opt}(\widetilde{I})$ to them. Consider the request $\widetilde{q}$:

1. If $\widetilde{q} \in \widetilde{Q}_{\mathtt{opt}} \cap \widetilde{Q}_{\mathtt{gdy}}$, by definition of $\widetilde{Q}_{\mathtt{gdy}}$ and $\widetilde{S}$, $\widetilde{q}$ will be served. Charge the requests ousted by $\widetilde{q}$ to H_1 at $(\mathtt{st}(\widetilde{q}), \mathtt{st}(\widetilde{q}) + \mathtt{sz}(\widetilde{q})]$. In addition, charge $\widetilde{q}$ to H_3 at $(\mathtt{st}(\widetilde{q}), \mathtt{st}(\widetilde{q}) + \mathtt{sz}(\widetilde{q})]$.

2. Otherwise, if $\widetilde{q} \in \widetilde{Q}_{\mathtt{opt}} - \widetilde{Q}_{\mathtt{gdy}}$ there are two sub-cases:

   (a) If $\widetilde{q}$ is allocated in $Plan(\widetilde{S}, \widetilde{I}, \widetilde{q})$, charge the requests ousted by $\widetilde{q}$ to H_2 $(\mathtt{st}(\widetilde{q}), \mathtt{st}(\widetilde{q}) + \mathtt{sz}(\widetilde{q})]$.

   (b) If $\widetilde{q}$ is not allocated, charge $\widetilde{q}$ to H_2 $(\mathtt{st}(\widetilde{q}), \mathtt{st}(\widetilde{q}) + \mathtt{sz}(\widetilde{q})]$.

It is easy to verify that the above describe a valid charging scheme. Thus we have

$$3 \cdot merit(\widetilde{S}(\widetilde{I})) \geq merit(\mathtt{opt}(\widetilde{I})),$$

as desired. **Q.E.D.**

7 General Lower Bound

From the previous section, we know that every greedy scheduler has competitive ratio that lies between 2 and 3. Are there schedulers outside the `Greedy` class with competitive ratio less than 2? We note a partial result in this direction: *every deterministic scheduler has competitive ratio at least* $2(2 - \sqrt{2}) > 1.17$.

In proof, consider this adversary: at time 0, the adversary releases two requests $q_0 := (0, 2, 1, 1)$ and $q_1 := (0, 1, 1, \sqrt{2} - 1)$. At time $t = 1$, let the residual size of q_0 be

s_0. If s_0 is less than $1/2$ then the request $q_2 := (1, 2, 1, 1)$ is released. Otherwise, no further requests will be released.

It may be verified that that any deterministic scheduler achieve a merit of at most $\frac{1}{2(2-\sqrt{2})}$ of the maximum possible.

Unfortunately, this lower bound of 1.17 leaves a wide gap from the current upper bound of 2. On the other hand, no simple variation of this adversary seems to give a better lower bound.

8 On the Number of Breakpoints and Multi-tasking

Both `FirstFit` and `EndFit` make $O(n)$ breakpoints where n is the number of requests. Does the number of breakpoints affect the performance of a scheduler? We give an alternative formulation of the scheduling problem which could be viewed as allowing an infinite number breakpoints. In this *multi-tasking* environment, several requests can be served concurrently, but each at a (possibly) different rate. However, in any time interval of size Δt, the total size of requests served within this interval must not exceed Δt. Alternatively, we allow "fractional service" where the total service at any moment sums to 1.

A concrete example of such a scheduler is `FirstEndFit`: It simulates `FirstFit` and `EndFit` concurrently and serves half of what `FirstFit` and `EndFit` would serve. That is, if `FirstFit` and `EndFit`will serve q and p respectively in the time slot $(s_0, t_0]$, then `FirstEndFit` will serve p and q concurrently but each at half the rate. We suspect that `FirstEndFit` is (3/2)-competitive.

Note that `FirstEndFit` can be also viewed as the following randomized scheduler under the original single-tasking setting:

1. Before receiving any request, it tosses a fair coin.

2. If the outcome is `head`, then it simulates `FirstFit`, otherwise it simulates `EndFit`.

Clearly, the expected merit gained by this randomized scheduler is same as the merit gained by `FirstEndFit`. In general, we can show that any randomized single-tasking scheduler is equivalent to a deterministic multi-tasking scheduler. We omit the details.

9 Conclusion

We have formulated a "level-of-service" scheduling problem that arises naturally in our thinwire visualization applications. This formulation is also useful in real-time systems where quality of jobs can be traded-off for time. We have derived several competitive algorithms in this setting. We continue to investigate the many interesting questions that are open. Besides sharpening the results in the paper, we pose the following directions for further work: (1) Find optimal schedulers which are not restricted to be on-line. (2) Study the problem with other measures of merit (instead of multiplicative weights in this paper). (3) Introduce a model of penalty for preemption. (4) Introduce randomization.

Acknowledgments

We thank Estie Arkin and Yi-Jen Chiang for discussions about the problem. The `EndFit` scheduler was proposed by Estie.

References

[1] S. Albers. Competitive online algorithms. BRICS Lecture Series LS-96-2, BRICS, Department of Computer Science, University of Aarhus, September 1996.

[2] S. Albers and J. Westbrook. A survey of self-organizing data structures. Research Report MPI-I-96-1-026, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, October 1996.

[3] E. Chang, C. Yap, and T.-J. Yen. Realtime visualization of large images over a thinwire. In *IEEE Visualization '97 (Late Breaking Hot Topics)*, pages 45–48, 1997. See CD proceedings of conference. Paper from `ftp://cs.nyu.edu/pub/local/yap/visual/thinwire.ps.gz`.

[4] E.-C. Chang. *Foveation Techniques and Scheduling Issues in Thinwire Visualization.* PhD thesis, Department of Computer Science, New York University, May 1998.

[5] J. A. Eoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. *Integer Programming and Combinatorial Optimization*, pages 404–414, 1996.

[6] R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. 5th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 302–311, 1994.

[7] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. In *Proc. 20th Annual ACM Symp. on Theory of Computing*, pages 322–333, 1988.

[8] L. A. McGeoch and D. D. Sleator, editors. *On-Line Algorithms.* DIMACS series in Discrete Mathematics and Theoretical Computer Science, volume 7. American Mathematical Society, 1992.

[9] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Comm. of the ACM*, 28(2):202–208, 1985.

[10] G. J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theor. Computer Science*, 130:5–16, 1994.

[11] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. 36th Annual Symp. on Foundations of Computer Science*, pages 374–382, 1995.