Preprint from Journal of Scheduling, 6(3):269–285, 2003.

Admission Control with Immediate Notification

Michael H. Goldwasser*
mhg@cs.luc.edu

Boris Kerbikov†
kerbikov@alumni.princeton.edu

Abstract

When admission control is used, an online scheduler chooses whether or not to complete each individual job successfully by its deadline. An important consideration is at what point in time the scheduler determines if a job request will be satisfied, and thus at what point the scheduler is able to provide notification to the job owner as to the fate of the request. In the loosest model, often seen in real-time systems, such a decision can be deferred up until the job's deadline passes. In the strictest model, more suitable for customer-based applications, a scheduler would be required to give notification at the instant that a job request arrives.

Unfortunately there seems to be little existing research which explicitly studies the effect of the notification model on the performance guarantees of a scheduler. We undertake such a study by re-examining a problem from the literature. Specifically, we study the effect of the notification model on the non-preemptive scheduling of a single resource in order to maximize utilization. At first glance, it appears severely more restrictive to compare a scheduler required to give immediate notification to one which need not give any notification. Yet we are able to present alternate algorithms which provide immediate notification, while matching most of the performance guarantees which are possible by schedulers which provide no such notification. In only one case are we able to give evidence that providing immediate notification may be more difficult.

1 Introduction

We consider the non-preemptive scheduling of a single resource in an online setting. Job requests arrive, with each request specifying the length of time for which the resource is needed as well as a deadline by which the job should be completed. The scheduler is not required to complete all job requests, yet the goal is to maximize the resource utilization. In such a setting, an important consideration is at what point in time the scheduler determines if a job request will be satisfied, and thus at what point the scheduler is able to provide notification to the job owner as to the fate of the request. The natural model for notification has varied greatly between application domains.

In traditional real-time systems, for example, admission control has been used for processing jobs with what are termed *firm* deadlines [17]. If a system issues a job request with a firm deadline this job does not necessarily need to be completed, however completing the job provides no utility if its deadline has passed. In this setting, it is quite natural to allow the scheduler to take a "wait-and-see" approach for each request. The scheduler is allowed to defer a final decision on whether or not to service the request up to the time when the deadline passes. From the scheduler's point of view, this is the loosest possible model for providing notification.

*Department of Computer Science, Loyola University, Chicago, 6525 N. Sheridan Rd., Chicago, IL 60626. Research completed while at Princeton University, supported by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center.

†Research completed while at Princeton University.

Conversely, admission control is routinely used in a wide array of customer-based applications, where job requests are submitted by individual customers to a company which may or may not be able to meet the requests by their given deadlines. Examples of such applications in daily life might include photo-finishing, dry cleaning, and package delivery. The problem has been motivated on a more industrial scale to model requests for an uninterrupted connection in a telecommunication system, used for the transmission of text, sound, images, video or other data. Unfortunately the loosest model for notification is inappropriate in these applications. If a customer has a job which "absolutely, positively" must be done by a certain deadline, it is unacceptable to entrust that job to a company that may wait until the deadline approaches before informing the customer that the request will not be completed. In this setting, we can consider the strictest possible model for notification. Immediately upon receiving a request, we can require that the scheduler either promise completion of the job or else outright reject it. In this way, if a request is rejected, the customer can still approach an alternate provider about servicing the job[1] .

Although these two views represent the extremes in modeling notification, it is possible to define various intermediate models by specifying at what point notification is required. For example, in an ATM network, the decision of whether to admit a request is not immediate, yet it is made during a specific connection setup period.

Unfortunately there seems to be little existing research which explicitly studies the effect of the notification model on the performance of a scheduler. Our goal is to undertake such a study, starting with this particular scheduling problem.

While receiving earlier notification can only be to the advantage of a customer, providing such notification can be nothing but a hindrance to the scheduler. Remarkably though, we are able to give alternate algorithms which provide immediate notification while matching most of the best possible performance bounds achieved by schedulers which provide no advanced notification [10, 12]. There does exist one case for which we are able to give evidence that providing immediate notification may indeed be strictly more difficult.

1.1 Definitions

Following the standard notation of Graham *et al.* [13], we consider a job J_i to be a triple of non-negative integers $\langle r_i, p_i, d_i \rangle$, where r_i is the arrival or release time of the job, p_i is the processing time, and d_i is the deadline for the job's completion. Job J_i is *available* at time t with respect to a schedule σ if $r_i \leq t \leq d_i - p_i$ and if job J_i has not been started in σ prior to time t. Additionally, we say that a job J_i has slack $s_i = d_i - p_i - r_i$, which is the amount of time between the job's arrival and the last possible time at which it could be started while still meeting its deadline. The *patience* of a problem instance is defined as $\kappa = \min_j(s_j/p_j)$, so that every job J_j with processing time p_j has a slack $s_j \geq \kappa \cdot p_j$ [12]. We let Δ denote the ratio between the largest and smallest processing times of an instance.

The *gain* of a schedule σ on instance $\mathcal{I}$ is defined equal to $\sum_{J_i \in \sigma} p_i$ and our goal is to maximize the gain. We assume that an online algorithm $\mathcal{A}$ has no knowledge of a job until the release time, at which point all job parameters are known. Furthermore, we assume that the algorithm has no a priori knowledge of the value of κ, however we will assume the algorithm has knowledge of the value of Δ. We will measure the performance of $\mathcal{A}$ by comparing its gain to the gain of an optimal offline algorithm, *opt*, that has a priori knowledge of all jobs when creating a schedule [7, 16, 21]. We say that a (deterministic) online algorithm $\mathcal{A}$ is *c-competitive* if $\text{gain}_{\text{opt}}(\mathcal{I}) \leq c \cdot \text{gain}_{\mathcal{A}}(\mathcal{I})$, for all input instances $\mathcal{I}$. When considering randomized online algorithms, the competitiveness compares

[1] Unfortunately, the authors' experiences have too often involved companies who give the appearance of immediate notification when accepting a request, yet later announce that the deadline will not be met.

the gain of the optimal schedule to the *expected* gain of a randomized algorithm. We assume that a worst case input for an algorithm is chosen by an *oblivious* adversary who must choose the entire sequence in advance [6, 20].

1.2 Our Results

Our primary results are algorithms which provide immediate notification while meeting the same bounds on performance as previous algorithms which do not provide any type of advanced notification. Specifically, we give algorithms providing immediate notification while matching the following competitiveness bounds given by previous algorithms without notification [10, 12]:

- For the case when all job lengths are equal, we present a deterministic algorithm which provides immediate notification. This algorithm is 2-competitive in general, and is $(1+\frac{1}{\lfloor\kappa\rfloor+1})$-competitive where $\kappa \geq 0$ is the patience of the instance.
- For the case when arbitrary job lengths are allowed and $\kappa > 0$ is the patience of an instance, we present a $(2+\frac{1}{\kappa})$-competitive, deterministic algorithm which provides immediate notification.
- For the case when arbitrary job lengths are allowed and $\Delta \geq 1$ is the ratio between the largest and smallest job lengths, we present a $(4\lceil\log_2 \Delta\rceil)$-competitive, randomized algorithm which provides immediately notification.

The first two of the above results are matched by existing lower bounds for the setting in which no advanced notification is required. These lower bounds trivially apply to the stricter setting of immediately notification and thus our results are tight. The third result improves, by a constant factor, the best known upper bound for the model with no advanced notification, and it is matched asymptotically by an $\Omega(\log \Delta)$ lower bound for that model [18].

Our secondary results concern the special case when all jobs have one of two distinct lengths. Although this special case has limited practical importance, it appears interesting theoretically. We are able to give some evidence that providing immediate notification may be strictly more difficult than providing no notification. Specifically, the best current results for the randomized competitiveness include a lower bound of 2 and an upper-bound of 4, in the model without notification [10, 18]. We make progress towards a separation between the competitiveness in the various notification models, giving the following results:

- We give a 4-competitive randomized algorithm which provides immediate notification.
- We give a 3-competitive randomized algorithm (without notification).
- We show that no randomized algorithm which provides immediate notification can be better than $\frac{7}{3}$-competitive.

Even with our improvements, we note that there is not yet an explicit separation between the two notification models, as the gaps between upper and lower bounds overlap.

2 Previous Work

Although understanding the effect of the notification model for admission control seems a clear priority, there has been very little mention of this issue in the standard literature. The only previous work we are able to find which explicitly studies the effect of the notification model on

performance involves a system for delivering video-on-demand [4]. In contract to our results, they show a striking difference in the competitiveness of that problem based on the exact model for notification.

A partial explanation for the lack of previous work focusing on the notification model is that the issue becomes moot in a non-preemptive model for which all jobs have zero slack. In such a setting, the scheduler is implicitly required to make an immediate decision of whether to begin a job or to reject the request.

Such is the case on the earliest work directly related to the problem we study. Lipton and Tomkins introduce a problem referred to to as *online interval scheduling*, in which all jobs request the *immediate* use of the resource [18]. When all job lengths are equal, a greedy online algorithm is guaranteed to find the optimal solution. Their model implicitly assumes that a scheduler does not have a priori knowledge of the value of Δ. When jobs have one of two distinct lengths, the authors provide a randomized 2-competitive algorithm and a matching lower bound. With arbitrarily many job lengths, the authors provide a randomized $O((\log \Delta)^{1+\epsilon})$-competitive algorithm, and a lower bound of $\omega(\log \Delta)$ for the competitiveness of any randomized online algorithm. Adapting their constructions to our model where the value of Δ is known results in lower bounds of $2 - \frac{1}{\Delta}$ with two distinct lengths and $\Omega(\log \Delta)$ with arbitrarily many lengths.

The model of Lipton and Tomkins was later generalized by Goldman, Parwatikar and Suri to a setting in which each job explicitly specifies a deadline of its choice [10]. As the scheduler may have flexibility in when to start a job, the choice of notification models becomes meaningful. The problem as defined by Goldman *et al.* implicitly corresponds to the setting in which no advanced notification is provided by the scheduler. In this setting, they provide a tight upper and lower bound of 2 for the deterministic competitiveness when all jobs have equal length. When all jobs have one of two lengths, they give a 4-competitive randomized algorithm, as compared to the lower bound of 2 from Lipton and Tomkins. Finally, when arbitrary job lengths are allowed, they provide a $6(\lceil \log_2 \Delta \rceil + 1)$-competitive, randomized algorithm, matching the $\Omega(\log \Delta)$ lower bound to within a constant factor.

Goldwasser considers the same setting as Goldman *et al.*, refining the analysis based on the introduction of κ, the patience of an instance, as an additional parameter [12]. When all jobs have equal length, Goldwasser proves that the same deterministic algorithm which Goldman *et al.* had shown to be 2-competitive is really $(1 + \frac{1}{\lfloor \kappa \rfloor + 1})$-competitive. Similarly, he proves that in the case where jobs have arbitrary lengths, a simple deterministic algorithm is $(2 + \frac{1}{\kappa})$-competitive. Matching lower bounds are given for both of these results.

Scheduling a single resource is a special case of the more general problem of call control in larger communication networks, where both admission control and routing are issues. The competitiveness of various call control models has been studied in both the non-preemptive [1, 2] and preemptive [3, 9] models; a more complete survey is given by Plotkin [19].

A technique of particular use in our work is given by Awerbuch, Bartal, Fiat and Rosén in solving some general call control problems on tree networks [2]. Though these problems do not closely relate to the problem we study, they introduce a technique for admission control termed *"Classify and Randomly Select"*. Jobs are classified into groups based on the individual job parameters such that jobs within a group are similar enough for a base algorithm to be competitive. Randomness is used to pick one particular class a priori, and then the overall algorithm runs the base algorithm on the selected group, while rejecting all jobs from other groups. The overall competitiveness increases by a factor equal to the number of groups. Our results will make use of the fact that if immediate notification can be provided by a base algorithm, then the overall "Classify and Randomly Select" technique can provide immediate notification.

```
ProcessArival(J)
    if Q = ∅ then
      ACCEPT J
      Q = {J}
      BeginJob( )
    else if Feasible(Q ∪ {J}, nextidle) then
      ACCEPT J
      Q = Q ∪ {J}
    else
      REJECT J

BeginJob( )
    Let J_i ∈ Q be the job with earliest deadline.
    nextidle ← current time + p_i
    Q ← Q − {J_i}
    Give resource to J_i.
    When J_i completes
      if Q ≠ ∅ then
        BeginJob( )

Feasible(S, t)
    Order all jobs of S by non-decreasing deadline {J_1, J_2, ..., J_s}
    Calculate lateststart ← min_{1≤j≤s}(d_j − Σ_{1≤i≤j} p_i)
    if t ≤ lateststart then
      return TRUE
    else
      return FALSE
```

Figure 1: The GREEDY-NOTIFY algorithm.

3 Providing Immediate Notification

A simple deterministic algorithm, denoted as GREEDY, was analyzed by Goldman *et al.* and by Goldwasser [10, 12]. This algorithm keeps all available jobs in a queue and whenever the resource becomes idle, it schedules the available job with the earliest deadline. As jobs waiting in the queue become infeasible they can be explicitly removed from the queue. Unfortunately, until this point a customer making a job request is given no notification as to whether or not the request will eventually be satisfied.

For this reason, we consider another natural greedy algorithm which provides immediate notification. The algorithm, GREEDY-NOTIFY, maintains a queue of *accepted* jobs. Whenever a new job arrives it accepts the job if and only if it can be feasibly scheduled along with all previously accepted jobs; otherwise it immediately rejects the job. The feasibility check is based on a classic result of Jackson stating that, in the absence of release times, the earliest due date ordering of jobs will produce a feasible schedule when one exists [15]. Specifically, if we are given a set $\mathcal{S}$ of *available* jobs, ordered $\{J_1, J_2, \ldots J_s\}$ by non-decreasing deadlines, the jobs can be feasibly scheduled starting

at time t if and only if the following inequality is satisfied,

$$t \leq \min_{1 \leq j \leq s} (d_j - \sum_{1 \leq i \leq j} p_i). \tag{1}$$

The complete algorithm is given in Figure 1.

3.1 An efficient implementation

The algorithm presentation of Figure 1 is clear, however it is not the most efficient, as the feasibility check requires that jobs be sorted according to deadline, and that a prefix sum of the processing times is computed. Naively this check requires $\Omega(|\mathcal{S}| \log |\mathcal{S}|)$ time, and the overall algorithm requires $\Omega(n^2 \log n)$ time in the worst case for an instance with n job requests. We can improve the running time by using a better data structure for maintaining the set $\mathcal{Q}$ and testing feasibility.

Theorem 1 *The algorithm* Greedy-Notify *can be implemented on an instance with n job requests, such that it runs in $O(\log n)$ time per job request and $O(n \log n)$ overall time.*

Proof: We will maintain the set $\mathcal{Q}$ ordered by non-decreasing deadlines using a red-black tree [5, 14]. We will further augment this tree to allow us to perform a feasibility test for the jobs of set $\mathcal{Q}$ given a starting time t. We make use of Theorem 15.1 of the text by Cormen, Leiserson and Rivest which states that extra fields can be added to nodes of a red-black tree, while still maintaining $O(\log n)$ running time for insertions and deletions, so long as the set of fields at a given node x can be computed solely from the information at x together with the values of the fields at the children of x [8]. In particular, each job $J_x \in \mathcal{Q}$ will be stored at some node x in which we maintain the following two additional fields:

- $total[x]$ – the sum of the processing times for all jobs stored in the subtree rooted at node x.
- $latest[x]$ – the latest possible starting time for a feasible scheduling containing only those jobs stored in the subtree rooted at node x.

The field $total[x]$ is maintained as the sum of p_x and the values of $total[left[x]]$ and $total[right[x]]$, in the cases where the respective child exists. The value of $latest[x]$ should equal the righthand side of Equation (1), when applied to the set of nodes in the subtree rooted at x. We claim this value can be calculated as follows,

$$latest[x] = \begin{cases} d_x - p_x & \text{if } x \text{ is a leaf} \\ \min(\quad latest[left[x]], & \text{otherwise} \\ \qquad d_x - p_x - total[left[x]], & \\ \qquad latest[right[x]] - p_x - total[left[x]]) & \end{cases}$$

When x is a leaf, the above is simply $d_x - p_x$. In general, an individual term from Equation (1) for a job J_i depends only on jobs in the set which have lessor or equal deadlines to that of J_i. Therefore, nodes in the left subtree of x have the same contribution for the tree rooted at x as they do for the tree rooted at $left[x]$. For x itself, the term is precisely is precisely $d_x - p_x - total[left[x]]$. Finally, the term associated with any job in the right subtree will be reduced exactly by the sum of p_x and $total[left[x]]$ because those jobs will surely have deadlines less than or equal to a job in the right subtree. Therefore, the minimum term contributed by a node in the right subtree of x will equal $latest[right[x]] - p_x - total[left[x]]$.

Based on these formulas, we can conclude that jobs can be inserted into or deleted from set $\mathcal{Q}$ in $O(\log |\mathcal{Q}|)$ time. Furthermore, the value of *latest* stored at the root represents the latest possible starting time for a feasible schedule containing all the jobs stored in the structure. This allows the condition (`Feasible`$(\mathcal{Q} \cup \{J\}, \mathit{nextidle})$) to be tested in $O(\log |\mathcal{Q}|)$ time, by inserting J into set $\mathcal{Q}$, comparing *nextidle* to the value of *latest*[*root*], and deleting J from the structure, in the case that it was infeasible.

Overall, we can be sure that each job accounts for at most one insertion into our structure, and thus at most one deletion. Since the size of the queue can never exceed n, we get an overall running time which is $O(n \log n)$. ∎

3.2 Equal Length Jobs

We begin by considering the case when all jobs have the same length. This is an important special case in its own right, for instance in ATM networks where packet sizes are all the same. In addition, we will use an algorithm designed for equal length jobs as a base algorithm for the more general case with multiple job lengths.

It would be convenient if we could show that the Greedy-Notify algorithm always produces the identical schedule as the Greedy algorithm considered by earlier researchers [10, 12]. Unfortunately this is not the case, as demonstrated by an instance with the following three jobs: $J_1 = \langle 0, 1, 1\rangle$, $J_2 = \langle \epsilon, 1, 2+\epsilon\rangle$, $J_3 = \langle 2\epsilon, 1, 2\rangle$. On this instance Greedy produces a schedule consisting of J_1 followed by J_3 whereas Greedy-Notify accepts both J_1 and J_2 and rejects J_3.

What we can prove is that in the special case where all job lengths are equal, Greedy-Notify and Greedy always produce schedules with the same gain. Specifically, we prove that the busy periods for the resource are exactly the same for schedules produced by the two algorithms. Therefore, all previous upper bounds for the performance of Greedy automatically apply to Greedy-Notify for this case.

Theorem 2 *When all job lengths are equal, if* Greedy *begins running a job J at some time t, then* Greedy-Notify *begins running a (possibly different) job at time t. Similarly, if* Greedy-Notify *begins running a job J at some time t, then* Greedy *begins running a job at time t.*

Proof: For simplicity, we assumed that both algorithms break ties lexicographically, when considering jobs with equivalent deadlines. We let σ denote the schedule produced by the Greedy algorithm and ν the schedule produced by the Greedy-Notify algorithm. For sake of contradiction, if the theorem is not true we let t be the earliest time at which one schedule starts running a job, J, while the other schedule does not start any job. We consider two cases, depending on whether σ starts job J at time t or ν starts job J at time t.

First we consider the case where σ starts J at time t. Inductively, we prove the existence, for any integer $k \geq 1$, of a sequence of jobs $\{J_{i_1}, J_{i_2}, J_{i_3}, \ldots, J_{i_k}\}$ and times $t_1 = t > t_2 > t_3 > \cdots > t_k$, such that

- At time t_1, ν is idle and σ begins job J_{i_1}.
- At time t_x, for all $2 \leq x \leq k$, ν begins job $J_{i_{x-1}}$ and σ begins job J_{i_x}.

The general framework is shown in Figure 2. As a base case, when $k = 1$, we let $t_1 = t$ and $J_{i_1} = J$. We have assumed that σ starts J at time t and since all jobs have equal length and t is defined as the earliest violation of the theorem, ν must remain idle at time t. For the inductive step, we assume a sequence for value k and prove the existence for $k+1$.

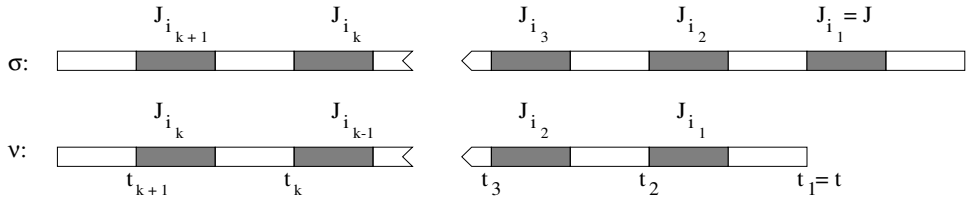

Figure 2: The first case of Theorem 2, in which σ runs J at time t.

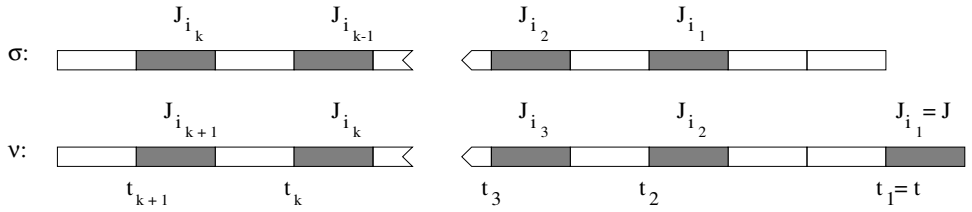

Figure 3: The second case of Theorem 2, in which ν runs J at time t.

We argue that job J_{i_k} was accepted by Greedy-Notify. To do so, we demonstrate a feasible schedule which includes J_{i_k} together with any previously accepted jobs of ν. As ν is idle at time t_1, it is possible to modify the schedule ν so that each job J_{i_x} is scheduled at time t_x rather than t_{x+1} for $1 \leq x \leq k-1$. The modified starting times respect the individual jobs' release times and deadlines, as witnessed by σ. Such a modification leaves an idle spot starting at time t_k which could be used to schedule J_{i_k}.

Both jobs J_{i_k} and $J_{i_{k-1}}$ arrived on or before time t_k as witnessed respectively by their placement in σ and ν. Both were available to σ at time t_k, yet Greedy chose to run J_{i_k} at that time based on their deadlines. From the preceding paragraph, we know that J_{i_k} had arrived and been accepted by ν on or before time t_k and yet Greedy-Notify begins running job $J_{i_{k-1}}$ at that time. This could only happen if it were the case that job J_{i_k} had run in ν prior to time t_k. Therefore we define time t_{k+1} as the time at which job J_{i_k} begins in ν. Since $t_{k+1} < t_k \leq t$, our assumption is that some job must also be started precisely at that time in σ. We call this job $J_{i_{k+1}}$, completing the induction.

Overall, the induction shows that an arbitrarily long such sequence of unique jobs can be created, since all times are unique. This contradicts the finiteness of a given instance, so it must be that our assumption is flawed regarding the existence of a time t when σ starts a job J yet ν idles.

For the second case in which the earliest violation involves ν starting job J at time t, we give a slightly simpler symmetric argument. Inductively, we prove the existence, for any integer $k \geq 1$, of a sequence of jobs $\{J_{i_1}, J_{i_2}, J_{i_3}, \ldots, J_{i_k}\}$ and times $t_1 = t > t_2 > t_3 > \cdots > t_k$, such that

- At time t_1, σ is idle and ν begins job J_{i_1}.
- At time t_x, for all $2 \leq x \leq k$, σ begins job $J_{i_{x-1}}$ and ν begins job J_{i_x}.

The general framework is shown in Figure 3. As a base case, when $k = 1$, we let $t_1 = t$ and $J_{i_1} = J$. We have assumed that ν starts J at time t and since all jobs have equal length and t is defined as the earliest violation of the theorem, σ must remain idle at time t. For the inductive step, we assume a sequence for value k and prove the existence for $k+1$.

From the inductive hypotheses, we see that both jobs J_{i_k} and $J_{i_{k-1}}$ arrived on or before time t_k as witnessed respectively by their placement in ν and σ. Furthermore, both jobs were accepted by Greedy-Notify and yet J_{i_k} was chosen to run at time t_k while job $J_{i_{k-1}}$ remained in the queue.

Since GREEDY chose to run $J_{i_{k-1}}$ at time t_k, we can conclude that job J_{i_k} must have already been completed in σ prior to time t_k. Therefore, we define time t_{k+1} as the time at which job J_{i_k} begins in σ. Since $t_{k+1} < t_k \leq t$, our assumption is that some job must also be started precisely at that time in ν. We call this job $J_{i_{k+1}}$, completing the induction. As in the first case, such an arbitrarily long sequence of unique jobs provides a contradiction, completing the proof. ∎

The following two corollaries follow trivially from Theorem 2 together with the previous analysis of GREEDY given by Goldman *et al.* [10] and Goldwasser [12], respectively.

Corollary 3 *When all jobs have the same length,* GREEDY-NOTIFY *is* 2*-competitive.*

Corollary 4 *When all jobs have the same length and a problem instance has a patience* $\kappa \geq 0$*,* GREEDY-NOTIFY *is* $(1 + \frac{1}{\lfloor \kappa \rfloor + 1})$*-competitive.*

3.3 Arbitrary Job Lengths

When we consider instances with arbitrary job lengths, not only is it the case that the schedules of GREEDY and GREEDY-NOTIFY may differ but the gain of the schedules may vary drastically. Consider an instance with three jobs $J_1 = \langle 0, 1, 2 \rangle$, $J_2 = \langle 0, 1, \Delta + 2 - \epsilon \rangle$ and $J_3 = \langle \epsilon, \Delta, \Delta + 1 \rangle$. The GREEDY-NOTIFY algorithm would achieve a gain of 2, as jobs J_1 and J_2 are accepted, however job J_3 is rejected as it is not possible to schedule all three jobs. Alternatively, the GREEDY algorithm would run job J_1 during $[0, 1)$, and at time 1 would choose to start J_3, achieving a gain of $1 + \Delta$.

Even so, we begin by considering the deterministic GREEDY-NOTIFY algorithm. Goldwasser defines an algorithm to be a *greedy-type* algorithm so long as the resource is never idle while a job is available [12]. We show that GREEDY-NOTIFY is indeed a greedy-type algorithm. If the resource is idle, all previously accepted jobs must have already completed. If a rejected job is available at such an idle time this contradicts the condition for rejection, as starting that job at this time provides a feasible schedule together with all previously accepted jobs. Based on this, the following two results follow trivially, as they were proven for any greedy-type algorithm by Goldwasser [12].

Theorem 5 *When jobs have arbitrary lengths,* GREEDY-NOTIFY *is* $(2 + \frac{1}{\kappa})$*-competitive.*

Lemma 6 *When jobs have arbitrary lengths with* Δ *equal to the ratio between the maximum and minimum lengths,* GREEDY-NOTIFY *is* $(2 + \Delta)$*-competitive.*

In order to design a competitive randomized algorithm, we apply Lemma 6 to provide a base algorithm for the "Classify and Randomly Select" technique, together with the following lemma given by Awerbuch *et al.* [2].

Lemma 7 *If the requests can be classified into a set,* $\mathcal{P}$*, of distinct groups, such that on each group* $P \in \mathcal{P}$ *a base algorithm is guaranteed to be c-competitive, then the "Classify and Randomly Select" technique provides a randomized algorithm which is* $c|\mathcal{P}|$*-competitive.*

Theorem 8 *When jobs have arbitrary lengths in the range* $[1, \Delta]$*, we can give a randomized algorithm with immediate notification which is* $(4\lceil \log_2 \Delta \rceil)$*-competitive.*

Proof: We can create $\lceil \log_2 \Delta \rceil$ distinct groups such that all jobs in a particular group are guaranteed to have lengths which are within a factor of two of each other. If we run GREEDY-NOTIFY on a set of jobs which have lengths within a factor of two of each other, Lemma 6 states that this algorithm will be 4-competitive. Using GREEDY-NOTIFY as a base algorithm for the "Classify and

Randomly Select" technique results in a $(4\lceil \log_2 \Delta \rceil)$-competitive randomized algorithm, according to Lemma 7. We note that because GREEDY-NOTIFY gives immediate notification and the "Classify and Randomly Select" technique can immediately reject those jobs not in the selected class, the resulting algorithm indeed provides immediate notification. ∎

Note that this upper bound is actually stronger than $6(\lceil \log_2 \Delta \rceil + 1)$-competitive algorithm provided by Goldman *et al.* for the case without any notification [10]. To be fair, in the case without any notification, Goldman *et al.* specifically sought an algorithm which did not rely on the "Classify and Randomly Select" technique, as this technique results in a large variance which may be unappealing in practice. It might be desirable to develop an algorithm with notification that does not rely on such a technique, but no such algorithm is evident.

4 The case with two distinct job lengths

We conclude by examining the randomized competitiveness of the case when all jobs have one of two known lengths. Without loss of generality we will assume all jobs have length either 1 or $\Delta > 1$, and we will refer to these groups as *small jobs* and *large jobs* respectively.

Unlike the previous cases, we give some evidence that the immediate notification model is strictly harder than the non-notification model for this setting. For the setting with immediate notification, we are able to provide an upper bound of 4 on the competitiveness, and a lower bound of $\frac{7}{3}$. In the previously studied model without any notification, we improve the best known upper bound to 3 opposite a previous lower bound of 2. Unfortunately, as these gaps overlap, we are unable to definitively separate the competitiveness of the two settings.

4.1 With immediate notification

We first consider the case of two jobs lengths in the setting where immediate notification is required. We provide a 4-competitive randomized algorithm, which matches the best previously known bound for the setting with no notification. Then we provide a lower bound of $\frac{7}{3}$ on the competitiveness of any randomized algorithm which provides immediate notification. This improves upon the lower bound of 2 which is known for the setting with no notification.

Theorem 9 *When jobs have one of two distinct lengths, there exists a 4-competitive, randomized algorithm which provides immediate notification.*

Proof: We again rely on the "Classify and Randomly Select" technique. This time, we create two groups, one consisting of small jobs and one consisting of large jobs. Since each group has only jobs of equal length, Corollary 3 guarantees that GREEDY-NOTIFY is 2-competitive when used as a base algorithm on each group. Thus, according to Lemma 7, the "Classify and Randomly Select" technique results in a randomized 4-competitive algorithm. As was the case with multiple length jobs, this algorithm provides immediate notification. ∎

Theorem 10 *When jobs have one of two distinct lengths, no randomized algorithm with notification can be c-competitive for $c < 7/3$.*

Proof: We fix an arbitrarily large Δ and define the following four problem instances, shown in Figure 4:

- Instance $\mathcal{I}_1$ consists of $J_1 = \langle 0, 1, 2+\epsilon \rangle$, $J_2 = \langle 2\epsilon, 1, 2 \rangle$;
- Instance $\mathcal{I}_2$ consists of $J_1 = \langle 0, 1, 2+\epsilon \rangle$, $J_2 = \langle \epsilon, 1, 2-\epsilon \rangle$;

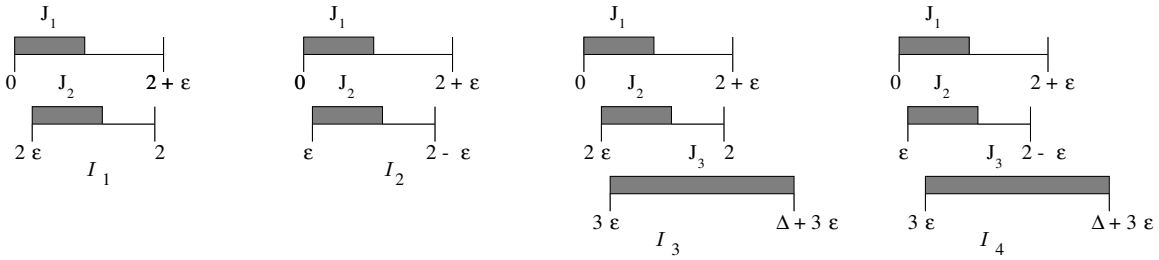

Figure 4: Four problem instances from proof of Theorem 10

		maximum possible gain given behavior			
instance	gain_{opt}	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$
$\mathcal{I}_1$	2	2	1	1	0
$\mathcal{I}_2$	2	1	2	1	0
$\mathcal{I}_3$	Δ	2	1	1	Δ
$\mathcal{I}_4$	Δ	1	2	1	Δ

Table 1: Summary of potential gains from proof of Theorem 10

- Instance $\mathcal{I}_3$ consists of $J_1 = \langle 0, 1, 2+\epsilon\rangle$, $J_2 = \langle 2\epsilon, 1, 2\rangle$, $J_3 = \langle 3\epsilon, \Delta, \Delta+3\epsilon\rangle$;
- Instance $\mathcal{I}_4$ consists of $J_1 = \langle 0, 1, 2+\epsilon\rangle$, $J_2 = \langle \epsilon, 1, 2-\epsilon\rangle$, $J_3 = \langle 3\epsilon, \Delta, \Delta+3\epsilon\rangle$.

Given any of the above four instances, we can classify a particular execution of an algorithm into one of exactly four disjoint behaviors:

- Behavior $\mathcal{A}$: J_1 was accepted and began processing at time $t = 0$;
- Behavior $\mathcal{B}$: J_1 was accepted, but the resource remained idle at time $t = 0$;
- Behavior $\mathcal{C}$: J_1 was rejected and J_2 was accepted;
- Behavior $\mathcal{D}$: both J_1 and J_2 were rejected.

Table 1 presents some observations based directly on the instance and behavior definitions.

Given any fixed randomized algorithm $\mathcal{R}$, we can let $\mathcal{P}_{\mathcal{A}}(\mathcal{I}_1)$ denote the probability that the algorithm produces behavior $\mathcal{A}$ when run on instance $\mathcal{I}_1$. We define similar notation for the probabilities of all other behaviors on all four instances. Since the four behaviors constitute a disjoint partition of all possible behaviors, we have for each $1 \le i \le 4$,

$$\mathcal{P}_{\mathcal{A}}(\mathcal{I}_i) + \mathcal{P}_{\mathcal{B}}(\mathcal{I}_i) + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_i) + \mathcal{P}_{\mathcal{D}}(\mathcal{I}_i) = 1 \tag{2}$$

At particular times, several of these instances are indistinguishable to an online algorithm, and so we claim:

$$\mathcal{P}_{\mathcal{A}}(\mathcal{I}_1) = \mathcal{P}_{\mathcal{A}}(\mathcal{I}_2) = \mathcal{P}_{\mathcal{A}}(\mathcal{I}_3) = \mathcal{P}_{\mathcal{A}}(\mathcal{I}_4) \tag{3}$$

$$\mathcal{P}_{\mathcal{B}}(\mathcal{I}_1) = \mathcal{P}_{\mathcal{B}}(\mathcal{I}_2) = \mathcal{P}_{\mathcal{B}}(\mathcal{I}_3) = \mathcal{P}_{\mathcal{B}}(\mathcal{I}_4) \tag{4}$$

$$\mathcal{P}_{\mathcal{D}}(\mathcal{I}_1) = \mathcal{P}_{\mathcal{D}}(\mathcal{I}_3) \tag{5}$$

$$\mathcal{P}_{\mathcal{D}}(\mathcal{I}_2) = \mathcal{P}_{\mathcal{D}}(\mathcal{I}_4) \tag{6}$$

Based on this, we simplify our notation, letting $\mathcal{P}_{\mathcal{A}} = \mathcal{P}_{\mathcal{A}}(\mathcal{I}_i)$ and $\mathcal{P}_{\mathcal{B}} = \mathcal{P}_{\mathcal{B}}(\mathcal{I}_i)$.

From Table 1, we infer the following lower bounds on the competitiveness of $\mathcal{R}$:

$$\begin{aligned}
\mathrm{comp}_{\mathcal{R}}(\mathcal{I}_1) &\geq \frac{2}{2\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_1)} && (7)\\
\mathrm{comp}_{\mathcal{R}}(\mathcal{I}_2) &\geq \frac{2}{\mathcal{P}_{\mathcal{A}} + 2\mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_2)} && (8)\\
\mathrm{comp}_{\mathcal{R}}(\mathcal{I}_3) &\geq \frac{\Delta}{2\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_3) + \Delta\mathcal{P}_{\mathcal{D}}(\mathcal{I}_3)} && (9)\\
\mathrm{comp}_{\mathcal{R}}(\mathcal{I}_4) &\geq \frac{\Delta}{\mathcal{P}_{\mathcal{A}} + 2\mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_4) + \Delta\mathcal{P}_{\mathcal{D}}(\mathcal{I}_4)} && (10)
\end{aligned}$$

We will show that it is not possible for $\mathcal{R}$ to have competitiveness strictly less than $\frac{7}{3}$ on each of these instances, proving the overall theorem. For the sake of contradiction, assume that the competitiveness is strictly less than $\frac{7}{3}$ on all four instances. Equation (9) assures us that $\mathcal{P}_{\mathcal{D}}(\mathcal{I}_3) > \frac{3}{7}$, as Δ was chosen to be arbitrarily large. Similarly Equation (10) assures us that $\mathcal{P}_{\mathcal{D}}(\mathcal{I}_4) > \frac{3}{7}$. Combining these bounds with the knowledge of Equations (2), (5) and (6) we see

$$\begin{aligned}
\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_1) &< \frac{4}{7} && (11)\\
\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_2) &< \frac{4}{7} && (12)
\end{aligned}$$

The assumption that $\mathrm{comp}_{\mathcal{R}}(\mathcal{I}_1) < \frac{7}{3}$ together with Equation (7) can be re-written to show $\frac{6}{7} < 2\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_1)$. Similarly, we can conclude from Equation (8) that $\frac{6}{7} < \mathcal{P}_{\mathcal{A}} + 2\mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_2)$. Adding these two inequalities, and referring to Equations (11) and (12), we conclude that

$$\begin{aligned}
\frac{12}{7} &< (2\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_1)) + (\mathcal{P}_{\mathcal{A}} + 2\mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_2)) = \\
&\quad (\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_1)) + (\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_2)) + \mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} \leq \\
&\quad 2(\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_1)) + (\mathcal{P}_{\mathcal{A}} + \mathcal{P}_{\mathcal{B}} + \mathcal{P}_{\mathcal{C}}(\mathcal{I}_2)) < \frac{4}{7} + \frac{4}{7} + \frac{4}{7} = \frac{12}{7}.
\end{aligned}$$

This contradicts our assumption that algorithm $\mathcal{R}$ was strictly less than $\frac{7}{3}$-competitive on all four instances. Therefore an oblivious adversary is always able to pick an instance on which $\mathcal{R}$ is at least $\frac{7}{3}$-competitive. ∎

4.2 Without notification

For the setting in which no notification is required, the upper bound of four is achieved with a randomized algorithm given by Goldman *et al.* [10]. Their algorithm, denoted Greedy-TwoLengths, is simple to describe. All available jobs are kept in two queues, $\mathcal{Q}_1$ for small jobs and $\mathcal{Q}_\Delta$ for large jobs. At any point the resource is idle, if there are any available large jobs, the algorithm runs the large job with earliest deadline. If there are no large jobs but there are small jobs available, then the algorithm flips a fair coin taking one of the following actions. With probability $\frac{1}{2}$, the algorithm runs the small job with earliest deadline. Otherwise, the algorithm immediately rejects the small job with earliest deadline, removing it from the queue, and it *virtually schedules* the resource for 1 unit of time blocking all other small jobs from running (but allowing a large job to begin if one arrives). The analysis of their algorithm is tight, as witnessed by an instance consisting of the two jobs $J_1 = \langle 0, 1, 3\rangle$ and $J_2 = \langle 0.5, 1, 1.5\rangle$. The optimal gain on this instance is 2, however the expected gain of Greedy-TwoLengths on this instance is $\frac{1}{2}$ as with probability $\frac{1}{2}$ it begins J_1

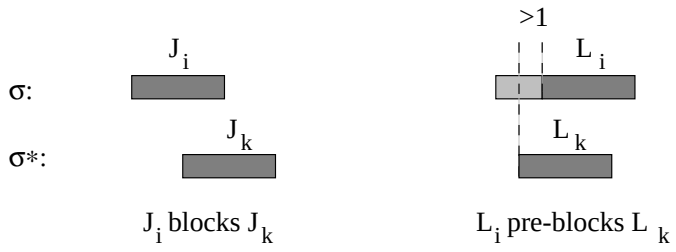

Figure 5: Definitions of *blocked* and *pre-blocked*.

at time 0, and with probability $\frac{1}{2}$ it rejects job J_0 while also virtually blocking the resource during $[0, 1)$. This provides a 4-competitive bound on this instance.

By making one minor modification to their algorithm, we are able to provide a 3-competitive algorithm. In the case that only small jobs exist, we choose to run the job with earliest deadline with probability $\frac{2}{3}$, thus rejecting the job and virtually scheduling the resource in the remaining $\frac{1}{3}$ of the time. We denote this algorithm Greedy-TwoLengths-Modified. Our analysis will be using a charging scheme which is adapted from that given by Goldman *et al.* We will be interested in a fixed instance $\mathcal{I}$ and we let σ^* denote an optimal schedule for the instance. For a particular execution of the randomized algorithm Greedy-TwoLengths-Modified on $\mathcal{I}$, we will let σ denote the schedule produced. For a job $J \in \sigma$, we let J^σ denote the time when job J has started running in σ, and similarly we define J^{σ^*} if $J \in \sigma^*$.

We begin with the following definitions, diagrammed in Figure 5. We say that a job $J_i \in \sigma$ *blocks* a job $J_k \in \sigma^*$ if $J_i^\sigma \leq J_k^{\sigma^*} < J_i^\sigma + p_i$. We say that a large job $L_i \in \sigma$ *pre-blocks* $L_k \in \sigma^*$ if L_i immediately follows the completion of a small job S in σ and if $L_k^{\sigma^*} < L_i^\sigma < L_k^{\sigma^*} + 1$.

For a single execution of the randomized algorithm Greedy-TwoLengths-Modified, we define the following scheme, mapping charge from those jobs completed in σ to those jobs completed in σ^*.

Assignment Rule 1: If small job S is scheduled in both σ and σ^*, S pays $\frac{1}{2}$ to itself.

Assignment Rule 2: If small job $S \in \sigma$ blocks $S_0 \in \sigma^*$, S pays $\frac{1}{2}$ to S_0.

Assignment Rule 3: If large job L is scheduled in both σ and σ^*, it pays $\frac{\Delta}{3}$ to itself.

Assignment Rule 4: Large job $L \in \sigma$ pays $\frac{1}{3}$ to each small job $S \in \sigma^*$ blocked by L.

Assignment Rule 5: If large job $L \in \sigma$ pre-blocks large job $L_0 \in \sigma^*$, L pays $\frac{\Delta}{3}$ to L_0.

Assignment Rule 6: If large job $L \in \sigma$ blocks large job $L_0 \in \sigma^*$, L pays $\frac{\Delta}{3}$ to L_0.

Assignment Rule 7: If $L \in \sigma$ has not yet paid out Δ, the remaining charge is paid to the same job as specified by Rule 6, if such a job exists.

Lemma 11 *For the given charging scheme, a job $J_i \in \sigma$ pays out at most p_i units of charge.*

Proof: A small job $S \in \sigma$ can pay out at most $\frac{1}{2}$ from Rule 1, and at most $\frac{1}{2}$ from Rule 2, as S can block at most one job. Therefore S pays out at most 1 unit.

For a large job $L \in \sigma$, we only need to consider Rules 3–6, as Rule 7 will never cause an overpayment. Rule 3 can account for payment of $\frac{\Delta}{3}$. We claim that Rules 4–6 can account for combined payment of at most $\frac{2\Delta}{3}$. If L pre-blocks a large job, then it can block at most one other

job. The maximum payout in such a case is when the blocked job is large, in which case $\frac{\Delta}{3}$ is paid out by each of Rules 5 and 6. If L does not pre-block a large job, it might block up to $\lceil\Delta\rceil$ other jobs, only one of which can be large. In such a situation, $\frac{\lceil\Delta\rceil-1}{3} < \frac{\Delta}{3}$ is paid out by Rule 4 and $\frac{\Delta}{3}$ is paid out by Rule 6. ∎

In considering the amount of charge collected by a particular job $J \in \sigma^*$, we must keep in mind that the charging scheme is a random process based on a particular execution of the randomized algorithm which leads to schedule σ. We can, however let $charge(J)$ be a random variable which represents the amount of charge which J collects. We can then let $\mathrm{E}[charge(J)]$ denote the *expected value* of the collected charge, where the expectation is taken over the randomness of the algorithm.

Lemma 12 *For a job $J_i \in \sigma^*$, $E[charge(J)] \geq \frac{p_i}{3}$.*

Proof: In bounding the expectation taken over all executions, we will partition the set of possible executions into distinct groups and we will individually analyze the conditional expectation taken over such a group $\mathcal{G}$ of executions. If we are able to show that $\mathrm{E}_{\mathcal{G}}[charge(J_i)] \geq \frac{p_i}{3}$ for each such group $\mathcal{G}$, then we can conclude the general bound. Such analysis does not rely on any explicit knowledge about the relative probability of an execution going to a particular such group $\mathcal{G}$.

We begin by considering a small job $S \in \sigma^*$, examining three distinct cases based on $t = S^{\sigma^*}$.

Case S1: Execution in which $\mathcal{Q}_1$ is empty at time t.
As witnessed by the placement of S in σ^*, the job has arrived on or before time t and is otherwise available to run. Since $\mathcal{Q}_1$ is empty, it must be the case that S was removed from the queue at an earlier time and either scheduled or virtually scheduled based on a coin flip. At such a point, there was an a priori probability of $\frac{2}{3}$ that S would have been scheduled and thus received $\frac{1}{2}$ of charge from Rule 1. Thus $\mathrm{E}[charge(S)] \geq \frac{2}{3} \cdot \frac{1}{2} = \frac{1}{3}$ in this case.

Case S2: Execution in which $\mathcal{Q}_1$ is not empty at time t, and a large job L begins to run at some point during the interval $(t-\Delta, t]$.
In this case L is blocking S, and so S will receive $\frac{1}{3}$ from Rule 4.

Case S3: Execution in which $\mathcal{Q}_1$ is not empty at time t, and a small job S_i receives the consideration of a coin flip at some point during the interval $(t-1, t]$.
S_i will be scheduled with probability $\frac{2}{3}$ and thus block S. Rule 2 would apply in this case, and thus $\mathrm{E}[charge(S)] \geq \frac{2}{3} \cdot \frac{1}{2} = \frac{1}{3}$.

We note that the above three cases indeed form a disjoint partition of the possible executions. In particular, if $\mathcal{Q}_1$ is not empty at time t, then either a large job is running, a small job is running or a small job is virtually scheduled.

We next consider a large job $L \in \sigma^*$, examining three distinct cases based on $t = L^{\sigma^*}$.

Case L1: Execution in which $\mathcal{Q}_\Delta$ is empty at time t.
As witnessed by the placement of L in σ^*, the job has arrived on or before time t and is otherwise available to run. Since $\mathcal{Q}_\Delta$ is empty, it must be the case that L was removed from the queue at an earlier time and scheduled. In this case it receives $\frac{\Delta}{3}$ based on Rule 3.

Case L2: Execution in which $\mathcal{Q}_\Delta$ is not empty at time t, and a large job L_i begins to run at some point during the interval $(t-\Delta, t]$.
In this case L_i will be blocking L, and so L will be paid $\frac{\Delta}{3}$ according to Rule 6.

Case L3: Execution in which $\mathcal{Q}_\Delta$ is not empty at time t, and a small job S receives the consideration of a coin flip at some point during the interval $(t-1, t]$.

Based on the algorithm, S would only be given this consideration at a time when $\mathcal{Q}_\Delta$ is empty, yet by time t, $\mathcal{Q}_\Delta$ is not empty. We let L_f denote the first large job which arrives after the point when S is considered. The significance of L_f is that it would be the large job which gets to run in the case that S were virtually scheduled. We further consider two sub-cases depending on the deadline of L_f.

Case L3a: $d_f < t + 2\Delta$.

We know that if S is virtually scheduled, then L_f gains the use of the resource when it arrives during $(t-1, t]$. We see that L_f blocks L which begins at time t in σ^* and thus L receives payment from Rules 6–7.

We claim that L will receive Δ from L_f. As L uses the resource in σ^* over the entire interval $[t, t+\Delta)$, it cannot be the case that L_f blocks any small jobs, and so no payment is made through Rule 4. As L_f begins after S is virtually scheduled and thus after an idle period, it cannot technically pre-block any job based on our definition, thus L_f makes no payment through Rule 5. Finally, we need only consider Rule 3. If it happens that $L_f = L$, then in any event, all Δ charge is received by L. Alternatively if $L_f \neq L$, we claim that L_f cannot possibly be scheduled in σ^*. It arrives strictly after time $t-1$ and has a deadline strictly less than $t + 2\Delta$. Therefore it cannot fit in σ^* either before or after job L which we already know runs throughout the interval $[t, t+\Delta)$.

Since S is virtually scheduled with probability $\frac{1}{3}$ and in this case, L receives Δ units, we can be sure that the expected charge received by L for this case is at least $\frac{\Delta}{3}$.

Case L3b: $d_f \geq t + 2\Delta$.

Although we do not know whether S becomes scheduled or virtually scheduled, we claim that L will receive $\frac{\Delta}{3}$ units of charge in either case. If S becomes virtually scheduled, then L_f will gain the resource when it arrives during $(t-1, t]$. In this case, L_f blocks L and so L receives $\frac{\Delta}{3}$ from Rule 6.

Alternatively, if S becomes scheduled, we claim that $\mathcal{Q}_\Delta$ will still be non-empty when S is completed, and therefore some large job will begin at that time, and such a large job exactly satisfies the conditions of pre-blocking L. For this reason, L will receive $\frac{\Delta}{3}$ from Rule 5.

Again, we conclude that these three cases form a disjoint partition of all possible executions. In particular, if $\mathcal{Q}_\Delta$ is not empty at time t a large job would begin running or else the resource must be currently scheduling an appropriate large or small job. ∎

Theorem 13 *In the case where jobs have one of two distinct lengths and no notification is required,* Greedy-TwoLengths-Modified *is a* 3*-competitive randomized algorithm.*

Proof: This result is a direct consequence of Lemmas 11 and 12. Given a particular instance $\mathcal{I}$ chosen by an oblivious adversary, we can analyze the expected performance of a randomized algorithm $\mathcal{R}$ versus the gain of the optimal schedule for that instance.

Although our charging scheme depends on the outcome of the randomized algorithm, we can bound the expected amount of charge paid out using Lemma 11, as follows,

$$\mathrm{E}[\text{overall charge paid out}] \leq \mathrm{E}[\sum_{J_i \in \sigma} p_i] = \mathrm{E}[\text{gain}_{\mathcal{R}}(\mathcal{I})].$$

Lemma 12 provides a bound comparing each job achieved in the optimal schedule to the expected amount of charge received by those jobs in our scheme. Summing this over all jobs in the optimal schedule, we see,

$$\text{gain}_{\text{opt}}(\mathcal{I}) = \sum_{J_i \in \sigma^*} p_i \leq 3 \cdot \left(\sum_{J_i \in \sigma^*} \text{E}[charge(J_i)] \right) = 3 \cdot \text{E}[\text{overall charge received}].$$

Finally, we use a simple rule of conservation of units of charge equating the expectations of charge received and charge paid. This allows us to conclude:

$$\frac{\text{gain}_{\text{opt}}(\mathcal{I})}{\text{E}[\text{gain}_{\mathcal{R}}(\mathcal{I})]} \leq \frac{3 \cdot \text{E}[\text{overall charge received}]}{\text{E}[\text{overall charge paid out}]} = 3.$$

∎

References

[1] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *Proc. 34th Symp. on Foundations of Computer Science*, pages 32–40, Palo Alto, California, Nov. 1993.

[2] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. Fifth Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 312–320, Arlington, Virginia, Jan. 1994.

[3] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schiber. Bandwidth allocation with preemption. In *Proc. 27th ACM Symp. on Theory of Computing*, pages 616–625, Las Vegas, Nevada, 29 May - 1 June 1995.

[4] A. Bar-Noy, J. A. Garay, A. Herzberg, and S. Aggarwal. Sharing video on demand. Manuscript, 1998. Presented at the *Workshop on Algorithmic Aspects of Communication*, Bologna, Italy, July 1997.

[5] R. Bayer. Symmetric binary B-trees: Data structure and maintenance. *Acta Informatica*, 1(4):290–306, 1972.

[6] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Widgerson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[7] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, 1998.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[9] J. Garey, I. Gopal, and S. Kutten. Efficient online call control algorithms. *Journal of Algorithms*, 23(1):180–194, 1997.

[10] S. Goldman, J. Parwatikar, and S. Suri. On-line scheduling with hard deadlines. *Journal of Algorithms*, 34(2):370–389, Feb. 2000.

[11] M. H. Goldwasser. Patience is a virtue: The effect of slack on competitiveness for admission control. In *Proc. Tenth Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 396–405, Baltimore, Maryland, Jan. 1999.

[12] M. H. Goldwasser. Patience is a virtue: The effect of slack on competitiveness for admission control. *Journal of Scheduling*, 2003. To appear. Earlier version appears as [11].

[13] R. Graham, E. L. Lawler, J. K. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. North Holland, 1979.

[14] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc. 19th Symp. on Foundations of Computer Science*, Lecture Notes in Computer Science, pages 8–21. Springer-Verlag, 1978.

[15] J. R. Jackson. Scheduling a production line to minimize maximum tardiness. Research Report 43, Managements Sciences Research Project, UCLA, January, 1955.

[16] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy paging. *Algorithmica*, 3(1):70–119, 1988.

[17] H. Kopetz. *Real-time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, 1997.

[18] R. Lipton and A. Tomkins. Online interval scheduling. In *Proc. Fifth Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 302–311, Arlington, Virginia, Jan. 1994.

[19] S. A. Plotkin. Competitive routing of virtual circuits in ATM networks. *IEEE J. Selected Areas in Communication*, 13(6):1128–1136, 1995.

[20] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38:683–707, 1994.

[21] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.