Report Number: WUCS-98-26

1998-01-01

On-line Scheduling with Hard Deadlines

Sally A. Goldman, Jyoti Parwatikar, and Subhash Suri

We study non-preemptive, online admission control in the hard deadline model: each job must be either serviced prior to its deadline, or be rejected. Our setting consists of a single resource that services an online sequence of jobs; each job has a length indicating the length of time for which it needs the resource, and a delay indicating the maximum time it can wait for the service to be started. The goal is to maximize total resource utilization. The jobs are non-preemptive and exclusive, meaning once a job begins, it runs to completion, and at most one job can... **Read complete abstract on page 2.**

On-line Scheduling with Hard Deadlines

Sally A. Goldman, Jyoti Parwatikar, and Subhash Suri

Complete Abstract:

We study non-preemptive, online admission control in the hard deadline model: each job must be either serviced prior to its deadline, or be rejected. Our setting consists of a single resource that services an online sequence of jobs; each job has a length indicating the length of time for which it needs the resource, and a delay indicating the maximum time it can wait for the service to be started. The goal is to maximize total resource utilization. The jobs are non-preemptive and exclusive, meaning once a job begins, it runs to completion, and at most one job can use the resource at any time. We obtain a series of results, under varying assumptions of job lengths and delays, which are summarized in the following table.

On-line Scheduling with Hard Deadlines

Sally A. Goldman, Jyoti Parwatikar and
Subhash Suri

WUCS-98-26

December 1998

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

On-line Scheduling with Hard Deadlines *

Sally A. Goldman†
Washington University
St. Louis, MO 63130-4899
sg@cs.wustl.edu

Jyoti Parwatikar‡
Washington University
St. Louis, MO 63130-4899
jp@cs.wustl.edu

Subhash Suri§
Washington University
St. Louis, MO 63130-4899
suri@cs.wustl.edu

WUCS-98-26

December 1998

Abstract

We study non-preemptive, online admission control in the hard deadline model: each job must be either serviced prior to its deadline, or be rejected. Our setting consists of a single resource that services an online sequence of jobs; each job has a *length* indicating the length of time for which it needs the resource, and a *delay* indicating the maximum time it can wait for the service to be started. The goal is to maximize total resource utilization. The jobs are non-preemptive and exclusive, meaning once a job begins, it runs to completion, and at most one job can use the resource at any time. We obtain a series of results, under varying assumptions of job lengths and delays, which are summarized in the following table.

*An earlier version appears in *Proceedings of the Workshop on Algorithms and Data Structures*, pages 258–271, August 1997.

† Supported in part by NSF Grant CCR-9110108 and NSF National Young Investigator grant CCR-9357707 with matching funds provided by Xerox PARC and WUTA.

‡ Research supported in part by Defense Advanced Research Projects Agency under contract DABT-63-95-C-0083.

§ Research supported in part by National Science Foundation Grant CCR-9501494

Job Lengths	Delay Type	Competitive Ratio
1	Arbitrary	2
1	Min. Delay 1	1.5
$\{1, k\}$	Arbitrary	4
$\{1, k\}$	Uniform	$\left(1 + \frac{\lceil k \rceil}{k}\right)$
$\{1, 2, 2^2, \ldots, 2^c\}$	Arbitrary	$3(c+1)$
$\{1, 2, 2^2, \ldots, 2^c\}$	Uniform	$2.5(c+1)$
$[1, 2^c]$	Arbitrary	$6(c+1)$
$[1, 2^c]$	"Uniform"	$5(c+1)$

1 Introduction

Online scheduling is an important problem area, with diverse applications. In this paper, we consider a scheduling framework where *jobs* have a *hard deadline*, meaning any job not serviced by its deadline is lost. Jobs arrive online, and must be scheduled non-preemptively. While much of the classical work on scheduling has been done in an offline setting, online scheduling is becoming especially important in high-speed network applications. The scheduling with hard deadlines models some interesting problems in providing Quality-of-Service (QoS) in shared packet-switched networks, as well as high bandwidth multimedia applications. For instance, in a packet-switched network such as the Internet, multiple traffic streams pass through a network of switching nodes (routers). Each node implements a *service discipline* (scheduling algorithm) for forwarding the incoming packets to outbound links. Due to the highly variable traffic rates (from few bits to several megabits per second) and packet sizes (from few bytes to several kilobytes), simple schedulers such as FIFO and round robin can fail miserably and fail to provide the bandwidth fairness and worst-case latency bounds. Packets in a latency-sensitive traffic stream such as video must be delivered within a small window, or else be dropped—the latency window corresponds to the maximum permissible delay at a router. (In these applications, delivering packets late has a worse effect than simply dropping the packets.) Similarly, in such emerging applications as video-on-demand, a useful model for requests is where users specify a window of time during which the delivery is acceptable.

Online scheduling in these applications can be used for admission control, accepting some requests and rejecting others with the goal of maximizing the total resource utilization. The requests or packets must be serviced *online* in these applications since the future arrival sequence is generally not known. We use competitive analysis[1] to measure the quality of our algorithms [13, 11, 6]; we consider both deterministic as well as randomized algorithms.

[1] We consider the *oblivious* competitive ratio in which the input sequence is selected independently of the random choices of the algorithm [5].

Job Lengths	Delay Type	Competitive Ratio
1	Arbitrary	2
1	Min. Delay 1	1.5
$\{1, k\}$	Arbitrary	4
$\{1, k\}$	Uniform	$\left(1 + \frac{\lceil k \rceil}{k}\right)$
$\{1, 2, 2^2, \ldots, 2^c\}$	Arbitrary	$3(c+1)$
$\{1, 2, 2^2, \ldots, 2^c\}$	Uniform	$2.5(c+1)$
$[1, 2^c]$	Arbitrary	$6(c+1)$
$[1, 2^c]$	"Uniform"	$5(c+1)$

Table 1: Summary of Results. We require that k is a real number greater than 1 and that c is a known integer. Uniform delay is a delay proportional to job length. However, in the last table entry "Uniform" refers to a delay proportional to $\lceil \lg |J| \rceil$ for job J.

1.1 Our Results

In earlier work, admission control algorithms have allowed preemption [8, 9], however, newer technologies seem to favor non-preemption. In particular, high speed networks based on packet switching technology (e.g. Asynchronous Transfer Mode) are *connection oriented*, meaning that resources are reserved during a call set up, and the overhead of this setup makes preemption highly undesirable. ATM networks are well-suited for latency-sensitive real-time traffic, such as video, voice, and multimedia. Consequently, there has been considerable recent work on non-preemptive admission control [1, 12, 3, 2, 4, 7]. Our work is most closely related to, and a generalization of, Lipton and Tomkins [12]. Our main results are summarized in Table 1. (By appropriate scaling, we can assume that the shortest job has length one. Uniform delay means that jobs of the same length have the same delay. The competitive ratios in all cases except for the first two are *expected*—the algorithms for the unit length jobs are deterministic, while all others are randomized.)

The unit length jobs may seem artificial but, in the networking context, they correspond to packet lengths, and therefore are well-suited to ATM where all packets have the same fixed size (53 bytes). Thus the special case of all jobs having the same length under the arbitrary delay model is of great interest. Also, it seems quite reasonable in this setting to require that there is a minimum delay. For other networking protocols and in the example of a video server, the jobs will have different lengths. As the table shows, for these settings we are able to get stronger results by enforcing uniform delays; this seems like a reasonable way to treat equal-length jobs.

When all jobs have unit length, we prove that the (deterministic) strategy of choosing

the *available* job with the earliest deadline first is strongly 2-competitive. (Note that this does not mean that jobs are scheduled in the earliest deadline order—a future job may have deadline sooner than an already scheduled job.) Strong-competitiveness means that no deterministic algorithm can do better in the worst-case. If *randomized* algorithms are permitted, then we can show a lower bound of 4/3 for the *expected* competitive ratio. When all the jobs have equal length and the *minimum* delay is at least the job length, then we prove that our greedy algorithm is 3/2-competitive.

When jobs have one of the two *known* lengths (1 and $k > 1$), we give a randomized algorithm that is 4-competitive. Additionally, if the delays are *uniform*, meaning that equal-length jobs have the same maximum delay, we give a randomized algorithm that is $\left(1 + \frac{\lceil k \rceil}{k}\right)$-competitive. For k an integer, this algorithm is strongly 2-competitive. This generalizes the result of Lipton and Tomkins [12], who consider jobs with *no delays*, and settles an open question of their paper. It is easy to see that there are instances (a large number of jobs arriving at roughly the same time, but each with a large allowable delay) where the *optimal solution with delays* is significantly better than that with no delays. While Feldmann et. al [7] show that there are worst-case inputs where delay does not help, we believe that in practice delay will help increase the resource utilization significantly. Finally, we extend our results to jobs of multiple lengths. These results are stated in Table 1 and are self-explanatory.

1.2 Previous Work

Our work is most closely related to the work of Lipton and Tomkins [12] and Awerbuch, Bartal, Fiat, and Rósen [3]. The paper of Lipton and Tomkins considers scheduling *without* delays, and we extend their work by considering several models of delays. If all delays are set to zero, our results achieve the same performance as Lipton and Tomkins. Finally, our results can be combined with the methods of Awerbuch et. al [3] to handle routing on a tree network or the situation in which the bandwidth of the requests can vary.

The paper by Awerbuch, Bartal, Fiat, and Rósen [3] considers admission control for tree networks. Again, no delays are allowed, meaning a request must be either scheduled immediately or rejected. They present a general technique called "classify-and-randomly-select" that randomly selects a bandwidth b, length ℓ, and benefit f, all of which are powers of 2. The algorithm then rejects all requests that do *not* have bandwidth between b and $2b$, length between ℓ and 2ℓ, and benefit between f and $2f$. They first give an $O(\log n)$-competitive algorithm for the case of a tree network of n nodes where all calls use the maximum bandwidth, have infinite length, and equal benefits. Then, by using the classify-and-randomly-select paradigm, they allow any of the other parameters to vary with a multiplicative increase of $\log \Delta$, where Δ is the ratio of the largest to smallest value for the parameter being varied. While the classify-and-randomly-select algorithm is simple and has provably good worst-

case performance, in practice, it seems unlikely that one would want an admission control algorithm that rejects all calls whose length is less than ℓ or greater than 2ℓ. In real-life, a rejected user is quite likely to immediately re-issue another request, forcing poor behavior for the classify-and-randomly-select algorithm. Allowing users to specify a maximum delay would eliminate such behavior. Our algorithm is also more natural for real-life use since it does not pre-select the lengths of the requests to accept.

Finally, Feldmann et. al [7] consider scheduling jobs *with delays* for a network that is a linear array of n nodes. They show that in some cases fixed length delays do not help. When delay is at most a constant multiple of the job length, they give request sequences where the competitive ratio between the amount of time the resource is used by an on-line algorithm with delays to the amount of time the resource is used by an off-line algorithm with no delays is $\Omega(\lg n)$. They also consider infinite delays, and try to optimize either the total completion time for all jobs, or the maximum delay between a job's arrival and its start of service. They present a $O(\log n)$-competitive greedy strategy for these measures on a n-node tree for requests of arbitrary bandwidth.

2 Preliminaries

Scheduling Model. A job (or call) J consists of a triple of positive real numbers $\langle a_J, |J|, w_J \rangle$, where a_J is the arrival time, $|J|$ is the length, and w_J is the maximum wait time for job J. That is, job J must be started during the interval $[a_J, a_J + w_J)$. A problem instance is a finite set $\mathcal{S}$ of jobs to be scheduled. We say that a schedule $\sigma \subseteq \mathcal{S}$ is *feasible* if no two jobs are running at the same time. We define the *gain* of a schedule σ to be $\sum_{J \in \sigma} |J|$. That is, it is the amount of time for which the resource is scheduled. (If clients are charged a rate per minute of usage, then the gain accurately reflects the profit of the schedule.

In the *uniform delay model*, the delay of each job is a function of the job's length. The only property of the uniform delay model used in our proofs is that jobs of the same length have the same delay. We also consider the *arbitrary delay model* in which each job can specify its own delay without any restrictions.

Method of Analysis. Our scheduling algorithms use randomization and thus we study their expected performance. We use $G_A(\mathcal{S})$ to denote the expected gain of algorithm A on a problem instance $\mathcal{S}$. The gain of algorithm A on a particular job $J \in \mathcal{S}$ is defined as $G_A(J) = \sum_\sigma \Pr[\sigma occurs] \Pr[J \in \sigma] \cdot |J|$, where the probability that σ occurs is with respect to the algorithm A, meaning the probability that A produces schedule σ given $\mathcal{S}$. We let $G_A(\mathcal{S}) = \sum_{J \in \mathcal{S}} G_A(J)$. When the algorithm being studied is clear, we just use $G(J)$ and $G(\mathcal{S})$. We use standard competitive analysis [13, 11, 6] to evaluate our algorithms.

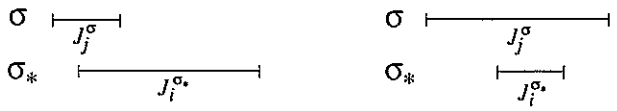

Figure 1: In the left drawing J_j^σ blocks $J_i^{\sigma*}$, and in the right one J_j^σ covers $J_i^{\sigma*}$. If $J_j^\sigma = J_i^{\sigma*}$ then by our definition of blocking, J_j blocks J_i.

Namely, let σ_* be an optimal solution (constructed by a computationally unbounded off-line algorithm). We say that algorithm A is *c-competitive* if $\forall \mathcal{S} : c \cdot G_A(\mathcal{S}) \geq |\sigma_*|$. An algorithm A is *strongly c-competitive* if it is c-competitive and there exists no c'-competitive algorithm for $c' < c$.

Additional Definitions. Each job in $\mathcal{S}$ is either (1) scheduled, (2) virtually scheduled, or (3) rejected. The job only runs if it is scheduled. A virtually scheduled job J does not run itself, but prevents any job of length $\leq |J|$ from running during a period of length $|J|$. Thus, virtually scheduling a job J holds the resource for a longer job (length $\geq 2|J|$) with a short wait time that may arrive during the interval when J is virtually scheduled.

A job can run at different times in σ and σ_*, so we use J_i^σ to denote J_i as run *or* virtually run in σ, and likewise use $J_i^{\sigma*}$ to denote J_i as run in σ_*. Abusing the notation slightly, we also use J_i^σ and $J_i^{\sigma*}$ as times at which J_i started in σ and σ_*, respectively. We say that a scheduled or virtually scheduled job J_j^σ *blocks* job $J_i^{\sigma*}$ if $J_j^\sigma \leq J_i^{\sigma*} < J_j^\sigma + |J_j|$. Job J_j^σ *covers* $J_i^{\sigma*}$ if $J_j^\sigma < J_i^{\sigma*}$ and $J_j^\sigma + |J_j| > J_i^{\sigma*} + |J_i|$. (See Figure 1.)

Basic Proof Structure. Given a schedule σ produced by algorithm A, we associate jobs in σ with jobs in the optimal schedule σ_*. For jobs $I \in \sigma$ and $J \in \sigma_*$, we let $A_\sigma(I, J)$ denote the portion of job I assigned to job J. The total gain associated with J for the schedule σ is defined as $gain(\sigma, J) = \sum_{I \in \sigma} A_\sigma(I, J)$. For $\mathcal{C}$ a set of possible schedules, we define $E_{\mathcal{C},A}[gain(J)] = \sum_{\sigma \in \mathcal{C}} \Pr[\sigma \text{ occurs}] \cdot gain(\sigma, J)$. When the algorithm A is clear from the context, we simply use $E_{\mathcal{C}}[gain(J)]$. Finally, when $\mathcal{C}$ is the set of all legal schedules for the jobs in $\mathcal{S}$, we use $E[gain(J)]$.

Our assignments for schedule σ will have the following two properties: (1) $\sum_{J \in \sigma_*} A_\sigma(I, J) \leq |I|$, and (2) for each $J \in \sigma_*$, $E[gain(J)] \geq |J|/r$ where $r > 0$. Property (1) is easily enforced. In order to achieve Property (2), we show that any schedule produced by our algorithm belongs to one of k cases: $\mathcal{C}_1, \ldots, \mathcal{C}_k$, where $\mathcal{C}_i$ occurs with probability p_i. We prove that for each job $J \in \sigma_*$ and each case $\mathcal{C}_i$, the following holds: $E_{\mathcal{C}_i}[gain(J)] \geq |J|/r$. It follows that

$$G_A(\mathcal{S}) \geq \sum_{J \in \sigma_*} \frac{|J|}{r} = \frac{G_{opt}(\mathcal{S})}{r}.$$

Properties (1) and (2) imply that our algorithm is r-competitive. Although a similar proof structure was used by Lipton and Tomkins, the assignment from jobs in $\mathcal{S}$ to the jobs in σ_* are more complicated in our case because we allow delays, and thus the assignments must depend on the particular schedule σ. In particular, a job can be run at different times (potentially overlapping) in different legal schedules (including σ and σ_*), which complicates the analysis significantly.

3 Unit Length Jobs

When no delays are allowed and all jobs have the same length, the greedy strategy of scheduling the jobs in the order of arrival is easily shown to be optimal. The problem becomes more complicated if arbitrary delays are allowed *even if all jobs have the same length.* The following theorem establishes lower bounds on the competitive ratio achievable by any online algorithm for this case. Without loss of generality, assume that all jobs have unit length.

Theorem 1 *Consider the task of scheduling jobs of unit length where each job can specify an arbitrary delay. No deterministic algorithm can be c-competitive for $c < 2$, and no randomized algorithm can be c-competitive for $c < 4/3$.*

Proof: Let scenario $\mathcal{S}_1$ consists of two jobs: J_1 arrives at time 0 and has a wait time of 1.25, and J_2 arrives at time .25 and has zero wait time. Scenario $\mathcal{S}_2$ also has two jobs: J_1 is same as before, but J_2 arrives at time 1 and has zero wait time. Suppose a scheduling algorithm A schedules J_1 at time $t = 0$ with probability p. (If A is deterministic, then $p \in \{0, 1\}$.) Then $G_A(\mathcal{S}_1) = p \cdot 1 + (1 - p) \cdot 2 = 2 - p$, and $G_A(\mathcal{S}_2) = p \cdot 2 + (1 - p) \cdot 1 = 1 + p$. The optimal schedule in each case has gain of 2. Thus, the competitive ratio for A is at least

$$\max\left(\frac{G_{opt}(\mathcal{S}_1)}{G_A(\mathcal{S}_1)}, \frac{G_{opt}(\mathcal{S}_2)}{G_A(\mathcal{S}_2)}\right) = \max\left(\frac{2}{2-p}, \frac{2}{1+p}\right) = \frac{2}{\min(2-p,\ 1+p)}.$$

If A is deterministic, we get $\min\{2 - p, 1 + p\} = 1$, since p is either 0 or 1, which implies that the competitive ratio of A is at least 2. If A is randomized, we get $\min\{2-p, 1+p\} \geq 3/2$, for $p = 1/2$, implying that 4/3 is a lower bound on the competitive ratio of A. □

Let Greedy denote the following greedy algorithm. At any time t, let $Q(t)$ be the set of all available jobs, meaning jobs J such that $a_J \leq t < a_J + w_J$. If the resource is free at time t and $Q(t) \neq \emptyset$, we schedule the job in $Q(t)$ that minimizes $(a_J + w_J) - t$; that is, the job with the smallest wait time remaining. We now prove that Greedy is strongly 2-competitive among all deterministic algorithms.

Theorem 2 *When jobs have unit lengths and arbitrary delays,* Greedy *is a strongly 2-competitive deterministic algorithm.*

Proof: For the upperbound, for each job $J \in \sigma$, assign $\frac{1}{2}|J|$ to the job blocked (if any) by J and the remainder of $|J|$ to itself. Let $J_j \in \sigma_*$. If $J_j \in \sigma$, then $G_{\mathsf{Greedy}}(J_j) \geq \frac{|J_j|}{2}$, since $\frac{|J_j|}{2}$ is assigned to itself. Suppose $J_j \notin \sigma_*$ then some job $J_i \in \sigma$ must block J_j and hence $A_\sigma(J_i, J_j) \geq \frac{|J_i|}{2} = \frac{|J_j|}{2}$. Thus Greedy is 2-competitive and hence by Theorem 1, it is strongly 2-competitive. □

We now consider the special case in which the minimum wait time is at least 1, the length of the jobs. This proof uses a different technique than that used in our other proofs.

Theorem 3 *When jobs have unit lengths and wait times of at least one,* Greedy *is a* $\frac{3}{2}$*-competitive deterministic algorithm.*

Proof: Consider the schedule σ produced by Greedy, and call the periods during which the resource is continuously in use the *busy periods* of σ. Label these periods as $\pi_1, \pi_2, \ldots, \pi_m$, and let b_i and e_i, respectively, denote the times at which π_i begins and ends. Partition the job sequence $\mathcal{S}$ into classes $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$, where $\mathcal{S}_i$ consists of exactly those jobs that *arrived* during the period $[b_i, e_i)$. Observe that Greedy schedules only jobs from $\mathcal{S}_i$ during π_i, and the job queue is empty at $t = e_i$.

We consider an arbitrary busy period π_i. For the purpose of analysis, we compress the schedule of the jobs in $\mathcal{S}_i$ produced by σ_*. Label the jobs in the schedule produced by σ_* on $\mathcal{S}_i$ as $J_1, J_2, \ldots, J_{|\mathcal{S}_i|}$ where J_k runs before J_l if $k < l$. We build the modified schedule σ_{*_i} in the following manner. Place J_1 in σ_{*_i} starting at b_i. For $1 < k \leq |\mathcal{S}_i|$, place J_k in σ_{*_i} starting as soon as J_{k-1} ends. In other words, we modify σ_{*_i} by backing up the jobs to remove any idle time. The resulting schedule may not be feasible since some jobs could start before their arrival time. However, this gives a lower bound for the performance of σ_* which is all we need. Using the modified schedule, the jobs are partitioned into three classes:

1. $L_i \subseteq \mathcal{S}_i$, the jobs scheduled in σ_{*_i} *after* the time e_i;
2. $R_i \subseteq \mathcal{S}_i$, jobs in σ_* that are *not* in σ;
3. jobs common to σ and σ_{*_i} during π_i.

Use the mnemonic "lazy" for L and "rush" for R: jobs in L_i have their deadlines past e_i, while all the jobs in R_i have their deadlines before e_i—otherwise the job queue of σ wouldn't be empty at e_i. Let l_i and r_i denote the cardinality of L_i and R_i, and let $k_i = |\pi_i|$ be the length of π_i in units of jobs. Then, it is clear that the total length of the jobs in $\mathcal{S}_i$ scheduled by σ_* cannot exceed either $k_i + l_i$ or $k_i + r_i$. Therefore, the competitive ratio of σ is

$$\min\left(\frac{k_i + l_i}{k_i}, \frac{k_i + r_i}{k_i}\right).$$

If $l_i \leq k_i/2$, then the competitive ratio during π_i is 3/2, and the proof is complete. Otherwise, we show that $r_i \leq k_i/2$ must hold. We claim that J_r (as run in σ_{*_i}) cannot overlap *two* lazy jobs in σ. Suppose it did, and let L and L' be the two jobs overlapping J_r. Since π_i is a busy period, jobs L and L' run consecutively, and let $t, t+1, t+2$ denote the time at the start of L, the end of L, and the end of L'. Since the job J_r has a *minimum delay* of 1, it must be eligible to run either at t or at $t+1$ (if it arrives during $(t, t+1)$, then it can run at $t+1$, and if arrives before t, then it can run at t). Since J_r is a rush job, its deadline expires before e_i, while both L and L' have their deadlines after e_i, we get a contradiction that *Greedy* schedules jobs in the order of earliest deadline first. A similar argument proves that no lazy job can overlap two rush jobs in σ_*. Because σ_{*_i} has no idle periods, the number of lazy jobs is at most $k_i - r_i$ since there is at most one per non-rush job in π_i. Thus, we have the inequality $l_i \leq k_i - r_i$. Since $l_i > k_i/2$, $k_i/2 < k_i - r_i$ yielding that $r_i < k_i/2$, which gives the desired upper bound of 3/2 on the competitive ratio. Since the ratio holds for all busy periods of σ, it holds for the entire schedule. □

4 Jobs of Two Lengths

We present a 4-competitive algorithm when each job can have one of two lengths, 1 and k (for any real $k > 1$). If the delays are uniform, rather than arbitrary, then we show that our algorithm is $\left(1 + \frac{\lceil k \rceil}{k}\right)$-competitive. We refer to the length-1 jobs as *short jobs*, and length-k jobs as *long jobs*. Let the short jobs be $S_1, \ldots, S_m$ *in their order of arrival*; similarly, let the long jobs be labeled $L_1, \ldots, L_n$, where L_i arrives after L_{i-1}. We use J_i to denote a job that could be of either length. Note that in this result, the competitive ratio does *not* depend on the ratio between the longest and shortest job.

Our scheduling algorithm, Schedule-Two-Lengths uses a queue Q_1 (respectively, Q_k) for short jobs (respectively, long jobs) that have arrived and are still waiting to be scheduled. Within each queue, the jobs are given priorities in decreasing order of the last time at which the job can be started. The basic idea is very simple. Whenever the resource is not in use (i.e. either unscheduled or virtually scheduled) and Q_k is not empty, we schedule the highest priority job from Q_k. If Q_k is empty and Q_1 is not empty then with probability 1/2 the short job of the highest priority from Q_1 is scheduled and otherwise it is virtually scheduled. Without loss of generality we assume that, if possible, two jobs in σ_* are scheduled so that the one with the earlier deadline is first. Our complete algorithm is show in Figure 2. If a short job is virtually scheduled, we remove it from Q_1 even if its wait time has not expired. It would seem that in practice leaving the job on Q_1 would only improve the resource utilization. In this case when we have jobs of two lengths, leaving jobs on the queue does not hurt the expected gain. However, when we consider jobs with arbitrary lengths, there are cases in

```
Schedule-TwoLengths
    Initialize Q_1 and Q_k to be the empty queues
    When job J arrives
        If |J| = 1 (i.e. J is a short job)
            If the resource is not scheduled nor virtually scheduled
                With probability 1/2 schedule J, otherwise virtually schedule J
            Else Place J in Q_1
        If |J| = k (i.e. J is a long job)
            If the resource is not scheduled then schedule J
            Else Place J in Q_k
    When a scheduled or virtually scheduled job finishes
        If Q_k is not empty
            Let J be the highest priority job in Q_k
            Remove J from Q_k and schedule J
        Else if Q_1 is not empty
            Let J be the highest priority job in Q_1
            Remove J from Q_1
            With probability 1/2 schedule J, otherwise virtually schedule J
```

Figure 2: Our admission control algorithm for when the jobs have lengths of 1 or k. The jobs in Q_1 (and also Q_k) are given a priority based on the latest time at which the job can start (with the highest priority given to the job that must start the soonest). We assume that jobs are removed from the queue when their wait time expires without explicitly including such checks in our pseudo-code.

which leaving jobs on the queue result in a worse expected gain.

We now prove that Schedule-TwoLengths is 4-competitive. Our proof is presented in a more general form than necessary to ease the transition to the case of arbitrary length jobs (which is given in the next section). For the schedule σ, we define the following graphs. For jobs of length ℓ (which is either 1 or k) we construct a graph $G_\ell = (V_\ell, E_\ell)$ as follows[2]. For each job of length ℓ there is a vertex in V_ℓ. If J_i^σ blocks $J_j^{\sigma *}$ for $|J_i| = |J_j| = \ell$, then the directed edge (J_i, J_j) is placed in E_ℓ. (Note that a virtually scheduled short job can block another short job.) Because each job can block only one other, the graph G_ℓ consists of a set of *chains* where a chain is a path $\langle (v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n) \rangle$. We refer to v_1 as the *head* of the chain, and v_n as the *tail*. A chain is a *singleton chain* if it contains a single vertex. The assignments $A_\sigma(\cdot, \cdot)$ are defined as follows:

Assignment 1: For $(S_i, S_j) \in E_1$ where S_i is scheduled in σ, let $A_\sigma(S_i, S_j) = A_\sigma(S_i, S_i) =$

[2]The graph G_ℓ is defined with respect to a schedule σ. For ease of exposition we use the notation G_ℓ versus $G_\ell(\sigma)$

1/2.

Assignment 2: Let L_i be a long job. For each short job $S_j \in \sigma_*$ covered or blocked by L_i^σ, let $A_\sigma(L_i, S_j) = 1/4$.

Assignment 3: For $(L_i, L_j) \in E_k$, let $A_\sigma(L_i, L_j) = k/2$.

Assignment 4: The remaining portion of L_i is assigned to itself.

Since a long job can cover at most k short jobs, it is easily seen that for each job $J \in \sigma$, $\sum_{J_j} A_\sigma(J, J_j) \leq |J|$. To prove that Schedule-Two-Lengths is 4-competitive we show that for all jobs J in σ_* the expected value of the assignment to J is at least $|J|/4$.

Theorem 4 Schedule-Two-Lengths *is 4-competitive in the model with arbitrary delays.*

Proof: We show that $E[gain(J)] \geq |J|/4$, for each $J \in \sigma_*$.

Case 1: J **is a short job** $S_j \in \sigma_*$. Let t be the time when S_j starts in σ_*. We partition the set of feasible schedules based on the following conditions:

$\mathcal{C}_{1a}$: A short job S_i is considered (removed from Q_1 and a coin flipped) during the interval $(t-1,\ t]$. With probability 1/2, S_i is scheduled thus blocking S_j. Since $A_\sigma(S_i, S_j) = 1/2$, $E_{\mathcal{C}_{1a}}[gain(S_j)] \geq (1/2)(1/2) = 1/4$.

$\mathcal{C}_{1b}$: A long job L_i is considered during the interval $(t-k,\ t]$. Since L_i runs with probability 1 and $A_\sigma(L_i, S_j) = 1/4$, $E_{\mathcal{C}_{1b}}[gain(S_j)] \geq 1/4$.

$\mathcal{C}_{1c}$: If neither of the above occur, then both queues must be empty at time t. However, S_j runs in σ_* at time t and thus it must have been removed from Q_1 because it was considered earlier. Thus with probability 1/2, S_j runs in which case $A_\sigma(S_j, S_j) = 1/2$. So $E_{\mathcal{C}_{1c}}[gain(S_j)] \geq (1/2)(1/2) = 1/4$.

Case 2: J **is a long job** $L_j \in \sigma_*$. Let t be the time when L_j starts in σ_*. We partition the set of schedules based on the following conditions:

$\mathcal{C}_{2a}$: A short job S is considered during the interval $(t-1,\ t]$. Q_k must be empty when S was considered, yet by time t, job L_j arrives. Let L be the first long job to arrive after S was considered. (Job L could be L_j or a different long job that arrives before L_j.) With probability 1/2, job S is virtually scheduled and thus L runs in σ when it arrives. Since $A_\sigma(L, L_j) = k/2$, $E_{\mathcal{C}_{2a}}[gain(S_j)] \geq (1/2)(k/2) = k/4$.

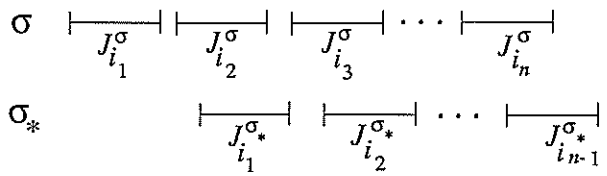

Figure 3: A backward chain. Note that jobs of other lengths can be interleaved within the chain. Also the jobs in σ could be scheduled or virtually scheduled.

$\mathcal{C}_{2b}$: A long job L_i is considered during the interval $(t-k,\ t]$. Since L_i runs with probability 1 and $A_\sigma(L_i, L_j) = k/2 \geq k/4$, $E_{\mathcal{C}_{2b}}[gain(L_j)] \geq \frac{k}{4}$.

$\mathcal{C}_{2c}$: If neither of the above occur, then both queues must be empty at time t. However, L_j runs in σ_* at time t and thus must have been removed from Q_k because it was considered earlier. The maximum amount of L_j is assigned to other jobs when L_j covers k short jobs ($k/4$ assigned) and blocks a long job ($k/2$ assigned). Thus $A_\sigma(L_j, L_j) \geq k - (k/2 + k/4) = k/4$.

Thus, we have shown that $E[gain(J)] \geq |J|/4$, for each $J \in \sigma_*$. This establishes $G_A(\mathcal{S}) \geq G_{opt}(\mathcal{S})/4$ as desired. □

We now consider the model of uniform delays: all jobs in the same class (short or long) have the same delay. In this case, we can assume that σ_* schedules jobs of the same class in the order of their arrival: if J and J' are two same length jobs in σ_*, then equal delay implies that they can be ordered by their arrival time. Observe that σ also schedules the jobs in this order. Ordering jobs (within its length class) by their arrival time allows us to take advantage of the following additional structure. We say a chain is a *backward chain* if each job blocks one that arrives before it. Thus in a backward chain jobs appear in the reverse order of arrival. (See Figure 3.)

Theorem 5 Schedule-Two-Lengths *is* $\left(1 + \frac{\lceil k \rceil}{k}\right)$*-competitive in the uniform delay model.*

Proof: Let $\alpha = \left(1 + \frac{\lceil k \rceil}{k}\right)$. The assignments $A_\sigma(\cdot, \cdot)$ are defined as follows:

Assignment 1: For $(S_i, S_j) \in E_1$ where S_i is scheduled in σ, $A_\sigma(S_i, S_j) = 1$.

Assignment 2: For each $S_j \in \sigma_*$ covered or blocked by the long job $L_i \in \sigma$, $A_\sigma(L_i, S_j) = 1/\alpha$.

Assignment 3: For $(L_i, L_j) \in E_k$, $A_\sigma(L_i, L_j) = k/\alpha$.

Assignment 4: For each job J, all unassigned portions of J are assigned to job J', where J' is the first job from σ_* in the chain of G_1 or G_k that includes J.

First, notice that $\sum_{J_j} A_\sigma(J, J_j) \leq |J|$, for each job $J \in \sigma$. Short jobs clearly satisfy the constraint. A long job can cover at most k short jobs and block one long job, and hence the total assigned value from Assignments 2 and 3 is $2k/\alpha$, which is at most k since $\alpha \geq 2$. We now prove that $E[gain(S_j)] \geq |S_j|/\alpha$ for each $S_j \in \sigma_*$.

Case 1: We consider when $S_j \in \sigma_*$ is a short job. As in the last proof, let t be the time $S_j^{\sigma_*}$.

$\mathcal{C}_{1a}$: A short job S_i is considered during the interval $(t-1,\ t]$. With probability $1/2$, S_i is scheduled and thus $A_\sigma(S_i, S_j) = 1$. Thus $E_{\mathcal{C}_{1a}}[gain(S_j)] = 1/2 \geq 1/\alpha$.

$\mathcal{C}_{1b}$: A long job L_i is considered during the interval $(t-k,\ t]$. Since $A_\sigma(L_i, S_j) = 1/\alpha$, $E_{\mathcal{C}_{1b}}[gain(S_j)] \geq 1/\alpha$.

$\mathcal{C}_{1c}$: Both queues are empty at time t and thus S_j was considered earlier. We now argue that S_j is the head of a backwards chain in G_1. If S_j does not block a short job in σ_* then S_j is a singleton chain. Suppose that S_j blocks $S_{j'}$. Then since $S_{j'}$ is before S_j in σ_* and there are uniform delays, we know that $S_{j'}$ arrived before S_j. Thus $S_{j'}$ must be considered in σ before S_j (or otherwise it would have been considered instead of S_j at time S_j^σ). So we must eventually reach a short job S_i that is considered in σ yet blocks no short job. Thus S_T is the tail of the chain with head S_j. Since only Assignment 4 applies for S_T, $A_\sigma(S_T, S_j) = 1$. S_T runs with probability $1/2$, and thus $E_{\mathcal{C}_{1c}}[gain(S_j)] \geq 1/2 \geq 1/\alpha$.

Case 2: We consider when $L_j \in \sigma_*$ is a long job. Let t be time $L_j^{\sigma_*}$.

$\mathcal{C}_{2a}$: A short job S is considered during the interval $(t-1,\ t]$, and thus Q_k was empty when S was considered. Let L be the first long job to arrive after S was considered. With probability $1/2$, job S is virtually scheduled and thus L runs in σ when it arrives. L cannot block or cover any short jobs since there is less than one unit between its arrival time and t. Since we have uniform delays, L's deadline must have passed by time t or otherwise σ_* would have scheduled it instead of L_j. Thus $L \notin \sigma_*$, and hence there is a chain in G_k with head L followed by L_j. Since $L \notin \sigma_*$, all of the unassigned portions of L are assigned to L_j. Thus $A_\sigma(L, L_j) = k$ and hence $E_{\mathcal{C}_{2a}}[gain(S_j)] \geq (1/2)k = k/2 \geq k/\alpha$.

$\mathcal{C}_{2b}$: A long job L_i is considered during $(t-k,\ t]$. So $E_{\mathcal{C}_{2b}}[gain(L_j)] \geq 1 \cdot A_\sigma(L_i, L_j) = k/\alpha$.

C_{2c}: Both queues are empty at time t and thus L_j was considered earlier. Using an argument as in Case 1c, it follows that L_j is the head of a backwards chain in G_k. Let L_T be the tail of the chain. The only portions of L_T not assigned to L_j are the $1/\alpha$ units assigned to each of the at most $\lceil k \rceil$ short jobs covered or blocked by L_T. Thus

$$E_{C_{2c}}[gain(S_j)] \geq 1 \cdot A_\sigma(L_T, L_j) \geq k - \frac{\lceil k \rceil}{\alpha} = k - \lceil k \rceil \left(\frac{k}{k + \lceil k \rceil} \right) = k \left(\frac{k}{k + \lceil k \rceil} \right) = \frac{k}{\alpha}.$$

□

The lower bound of 2 given by Lipton and Tomkins [12] holds here (since all delays could be zero), thus we immediately obtain the following corollary.

Corollary 6 Schedule-Two-Lengths *is strongly 2-competitive in the uniform delay model when the ratio, k, between the length of a long job and the length of a short job is an integer.*

5 Arbitrary Length Jobs

In this section we consider jobs that can have any length, but the maximum length of a job is known to the algorithm; as usual, we assume that the shortest job has length 1. We first consider the case when job lengths are powers of 2, namely, $1, 2, 4, \ldots, 2^c$, for some known constant c. While our algorithm is similar to the Marriage Algorithm of Lipton and Tomkins [12], our analysis is completely different. Note that the classify-and-randomly-select paradigm [3] in this case would just pick one randomly selected length to schedule. Our approach, in contrast, seems much more reasonable in practice: though there is a bias in favor of longer jobs, shorter length jobs also have a chance of being scheduled. The following lower bound was shown by Lipton and Tomkins [12].

Theorem 7 *[12] Let $\Delta = 2^c$ be the ratio between the longest and shortest lengths. There is a lower bound of $\Omega(\log \Delta) = \Omega(c)$ for the randomized competitive ratio in the model with no delays.*

Following Schedule-Two-Lengths we maintain a separate queue for different length jobs. We use Q_ℓ to denote the queue for jobs of length 2^ℓ. We favor the longer jobs in that (1) the probability of scheduling a job of length 2^ℓ is $1/(c+1-\ell)$ so the longer jobs are more likely to be scheduled, and (2) whenever the resource becomes available we schedule or virtually schedule the longest job available. During the period a job J is virtually scheduled,

```
Schedule-With-General-Delays
    Initialize Q_0, ..., Q_c to be the empty queues
    When job J arrives
        Let 2^ℓ be the length of J (for 0 ≤ ℓ ≤ c)
        If the resource is not scheduled, or if J' is virtually scheduled where |J'| ≤ 2^(ℓ-1)
            With probability 1/(c+1-ℓ) schedule J, otherwise virtually schedule J
        Else Place J in Q_ℓ
    When a scheduled or virtually scheduled job finishes
        Let Q_ℓ be the non empty queue for the largest ℓ possible
            Let J be the highest priority job in Q_ℓ
            Remove J from Q_ℓ
            With probability 1/(c+1-ℓ) schedule J, otherwise virtually schedule J
```

Figure 4: Our admission control algorithm for when the jobs have lengths of $1, 2, 4, \ldots, 2^{c-1}$, or 2^c for a known constant c.

it prevents jobs of the same length or shorter jobs from running, but not longer ones. Our complete algorithm is shown in Figure 4.[3]

As in Schedule-Two-Lengths, if a job is virtually scheduled, we remove it from the queue even if its wait time has not expired. However, in this case, leaving a job on the queue may result in a worse expected gain. For example, a long job with a large delay arrives when the resource is free and a succession of shorter jobs with small delays arrive soon after. As long as the larger job is virtually scheduled, the resource is not in use and we do not consider the shorter jobs. If the shorter jobs have a very small delay, they time out before we get an opportunity to consider them. During the period this job is virtually scheduled, the optimal algorithm will schedule both the long job and a series of short jobs. Intuitively, if we do not remove the large job from the queue after it has been considered once and it is virtually scheduled at least once, our expected gain decreases since we never give the short jobs a chance to run.

We now prove that this algorithm is $3(c+1)$-competitive. As in Schedule-Two-Lengths we use a graph for jobs of each length. Instead of using $G_{2^\ell} = (V_{2^\ell}, E_{2^\ell})$ for the graph corresponding to jobs of length 2^ℓ we use $G_\ell = (V_\ell, E_\ell)$ for $0 \leq \ell \leq c$. For each job of length 2^ℓ there is a vertex in V_ℓ. If a scheduled or virtually scheduled job J_i^σ blocks $J_j^{\sigma *}$ where $|J_i| = |J_j| = 2^\ell$ then the directed edge (J_i, J_j) is placed in E_ℓ. Again each graph G_ℓ consists of set of chains. For a given schedule σ, we make the following assignments based on G_ℓ (for $0 \leq \ell \leq c$). Unless otherwise given, the length of job J_i is 2^{ℓ_i}.

[3]For ease of exposition, we again assume that jobs are removed from the queue when their wait time expires without explicitly including such checks in our pseudo-code.

Assignment 1: If job J_i covers or blocks J_j (so $\ell_i \geq \ell_j$), then $A_\sigma(J_i, J_j) = 2^{\ell_j}/3$.

Assignment 2: The remaining portion of J_i is assigned to itself.

Since the sum of the lengths of the jobs covered by J_i is at most 2^{ℓ_i}, the maximum amount assigned from J_i in Assignment 1 is $2^{\ell_i}/3$. Thus clearly, $\sum_{J_j \in \sigma_*} A_\sigma(J_i, J_j) \leq |J_i|$, for each $J_i \in \sigma$.

Theorem 8 Schedule-General-Delays *is* $3(c+1)$*-competitive if job lengths are in the set* $\{1, 2, 4, \ldots, 2^c\}$*, for some known constant* c*.*

Proof: It suffices to prove that $E[gain(J)] \geq \frac{|J|}{3(c+1)}$ for each $J \in \sigma_*$. Let J_i be the last job considered in σ before time $t = J_j^{\sigma_*}$. We consider the following cases.

$\mathcal{C}_1$: $\ell_i < \ell_j$ **and** J_i **is considered during the interval** $(t - 2^{\ell_i},\ t]$. Let J be the first job of length ℓ_j to arrive after J_i is considered. (Job J could be J_j or a different job of the same length that arrives before J_j.) In the worst case, our algorithm might consider a job of length $\ell_i, \ell_i + 1, \ldots, \ell_j - 1$ before considering job J. In order for J to run, the other jobs considered (at most one per length) must be virtually scheduled and then J must be scheduled. Thus:

$$\begin{aligned}
\text{Prob } J \text{ scheduled} &\geq \left(1 - \frac{1}{c+1-\ell_i}\right)\left(1 - \frac{1}{c+1-(\ell_i+1)}\right)\cdots\left(1 - \frac{1}{c+1-(\ell_j-1)}\right)\left(\frac{1}{c+1-\ell_j}\right) \\
&= \left(\frac{c-\ell_i}{c+1-\ell_i}\right)\left(\frac{c-1-\ell_i}{c-\ell_i}\right)\cdots\left(\frac{c+1-\ell_j}{c+2-\ell_j}\right)\left(\frac{1}{c+1-\ell_j}\right) \\
&= \frac{1}{c+1-\ell_i} \geq \frac{1}{c+1}
\end{aligned}$$

The above argument can be applied to show that any job considered would be scheduled with probability at least $1/(c+1)$. Since J runs with probability at least $1/(c+1)$ and $A_\sigma(J, J_j) = 2^{\ell_j}/3$, $E_{\mathcal{C}_1}[gain(J_j)] \geq \frac{2^{\ell_j}}{3(c+1)}$.

$\mathcal{C}_2$: $\ell_i \geq \ell_j$ **and** J_i **is considered during the interval** $(t - 2^{\ell_i},\ t]$. Since the probability that J_i runs is at least $1/(c+1)$, and $A_\sigma(J_i, J_j) = 2^{\ell_i}/3$, $E_{\mathcal{C}_2}[gain(J_j)] \geq \frac{2^{\ell_i}}{3(c+1)} \geq \frac{2^{\ell_j}}{3(c+1)}$.

$\mathcal{C}_3$: **All queues are empty at time** t. This case must occur if neither of the above two cases do. Since Q_j is empty, J_j was considered earlier and ran with probability at least $1/(c+1)$. By Assignment 1, at most $2^{\ell_j}/3$ of J_j is assigned to jobs it covers and at most $2^{\ell_j}/3$ of J_j is assigned to the job it blocks (if any). Thus $A_\sigma(J_j, J_j) \geq 2^{\ell_j} - (2^{\ell_j}/3 + 2^{\ell_j}/3) = 2^{\ell_j}/3$. Thus $E_{\mathcal{C}_3}[gain(J_j)] \geq \frac{2^{\ell_i}}{3(c+1)}$.

For each $J \in \sigma_*$, we have $E[gain(J)] \geq \frac{|J|}{3(c+1)}$, thus the theorem follows. □

Extending this result to the case in which the jobs have lengths between 1 and 2^c we obtain the following corollary.

Corollary 9 Schedule-General-Delays *is* $6(c+1)$*-competitive if jobs have lengths between* 1 *and* 2^c*, for some known constant* c*.*

Proof: The only modification needed in Schedule-General-Delays is to treat job J like a job of length $2^{\lceil \lg |J| \rceil}$. Two key observations used in the proof are that (1) each job $J \in \sigma$ has true length that is at least 1/2 of $2^{\lceil \lg |J| \rceil}$, and (2) if job $J \in \sigma_*$ is shorter than $2^{\lceil \lg |J| \rceil}$ then we can apply the excess that has been assigned to J to any other jobs (or portions thereof) that run in σ_* between J^{σ_*} and $J^{\sigma_*} + 2^{\lceil \lg |J| \rceil}$. □

We now consider the special case when there are uniform delays. Note that each graph G_ℓ consists of set of chains where each chain is a forward chain, backward chain, or singleton chain.

Theorem 10 Schedule-General-Delays *is* $2.5(c+1)$*-competitive in the uniform delay model when jobs have lengths in the set* $\{1, 2, 4, \ldots, 2^c\}$*, for some known constant* c*.*

Proof: We use the following assignments:

Assignment 1: If job J_i covers or blocks J_j (so $\ell_i \geq \ell_j$), then $A_\sigma(J_i, J_j) = 2^{\ell_j}/2.5$.

Assignment 2: For each job J_i, all unassigned portions of J_i are assigned to job J', where J' is the first job from σ_* in the chain G_{ℓ_i}.

Let J_i be the last job considered in σ before time $t = J_j^{\sigma_*}$. We consider the following cases.

$\mathcal{C}_1$: $\ell_i < \ell_j$ **and** J_i **is considered during the interval** $(t - 2^{\ell_i}, t]$. As before, let J be the first job of length ℓ_j to arrive after J_i is considered. Using the same argument as in case $\mathcal{C}_1$ in the proof of Theorem 8, we get that the probability that J runs is at least $1/(c+1)$. Since there are uniform delays J's deadline must have expired by time t (or σ_* would have scheduled it). Thus $J \notin \sigma_*$, and hence J_j is in a chain with head J followed by J_j. So, in Assignment 2, all excess from J is assigned to J_j. Finally, since J can cover jobs of length at most $2^{\ell_i} \leq 2^{\ell_j - 1}$ and blocks J_j, it follows that $A_\sigma(J, J_j) \geq 2^{\ell_j} - \frac{2^{\ell_j - 1}}{2.5} = 2^{\ell_j}/1.25$. Thus $E_{\mathcal{C}_1}[gain(J_j)] \geq \frac{2^{\ell_j}}{1.25(c+1)} \geq \frac{2^{\ell_j}}{2.5(c+1)}$.

$\mathcal{C}_2$: $\ell_i \geq \ell_j$ **and** J_i **is considered during the interval** $(t - 2^{\ell_i}, t]$. Since the probability that J_i runs is at least $1/(c+1)$ and $A_\sigma(J_i, J_j) = 2^{\ell_j}/2.5$, $E_{\mathcal{C}_2}[gain(J_j)] \geq \frac{2^{\ell_i}}{2.5(c+1)} \geq \frac{2^{\ell_j}}{2.5(c+1)}$.

$\mathcal{C}_3$: **All queues are empty at time t.** Thus J_j must be considered earlier. As when we considered uniform delays in the two job case, it can be shown that J_j is the head of a singleton chain with tail J_T. The maximum amount from J_T assigned to other jobs occurs when it covers jobs of lengths 2^{ℓ_j} and blocks a job of length 2^{ℓ_j-1}. Thus $A_\sigma(J_T, J_j) \geq 2^{\ell_j} - \left(\frac{2^{\ell_j}-2^{\ell_j-1}}{2.5}\right) = \frac{2\cdot 2^{\ell_j}}{5}$, and hence $E_{\mathcal{C}_3}[gain(J_j)] \geq \frac{2\cdot 2^{\ell_j}}{5(c+1)} = \frac{2^{\ell_j}}{2.5(c+1)}$. □

Corollary 11 Schedule-General-Delays *is* $5(c+1)$*-competitive when scheduling jobs between lengths* 1 *and* 2^c *for some known constant* c *where the delay for each job* $J \in \mathcal{S}$ *is a function of* $\lceil \lg |J| \rceil$.

6 Concluding Remarks

We have presented upper and lower bounds on the competitive ratio for non-preemptive, online admission control in the hard deadline model: each job must be either serviced prior to its deadline, or be rejected. Our results are summarized in Table 1.

Recently, additional results along the line of Theorem 3 in which you obtain stronger competitive bounds by imposing a minimum delay have been obtained by Goldwasser [10]. In particular, he considers the restriction in which each job J must have a delay (slack) at least $\kappa \cdot \|J\|$ for some constant $\kappa > 0$. He gives a simple greedy algorithm that is $(2 + \frac{1}{\kappa})$-competitive even when arbitrary job lengths are allowed and gives lower bounds showing that this is the best possible result for a deterministic algorithm even if all jobs have one of three distinct lengths. In the special case where all jobs have the same length, he generalizes a previous bound of 2 for the deterministic competitiveness with arbitrary slacks, showing that the competitiveness for any $\kappa \geq 0$ is exactly $1 + \frac{1}{\lfloor \kappa \rfloor + 1}$. He also gives tight bounds for the case where jobs have one of two distinct lengths.

There are many interesting open questions raised by our work. In the model of unit length jobs with arbitrary delays, we have a lower bound (for randomized algorithms) of 4/3 on the competitive ration. Yet the best algorithm we've given is Greedy, which is a deterministic algorithm that is 2-competitive. Can randomization be used to obtain a better result? Can the lower bound be improved? Another interesting open question is to study the off-line problem of scheduling unit length jobs with arbitrary delays in the hard deadline model: Is there a polynomial time algorithm to find an optimal solution, or is this a NP-hard problem? While there is significant work on off-line scheduling, none addresses the hard deadline model.

While we have lower bounds to demonstrate that the competitive ratios of Schedule-TwoLengths and Schedule-With-General-Delays are asymptotically tight (in both the uniform and arbitrary delay models), the constants may not be tight. Thus another open problem is

to try to develop matching upper and lower bounds on the competitive ratio for the various settings we considered.

Finally, another direction of study is to associate an additional *payoff* parameter with each job, which is the amount that a job is willing to pay for being scheduled. (In our current work, the implicit payoff was always equal to the length of the job.) Under this model a job that has a very short delay could provide a high payment to increase the chance that it is scheduled.

Acknowledgments

We thank Ben Gum who participated in the early discussions of this work. We also thank Michael Goldwasser and Eric Torng for many helpful discussions on this work. Finally, we thank David Mathias and Michael Goldwasser for their comments on an earlier draft of this paper.

References

[1] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Proc. of the 34th Ann. Symp. on Foundations of Comp. Sci.*, pp. 32–41. IEEE, Computer Society Press, Los Alamitos, CA, 1993.

[2] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown durations. In *Proc. of the 5th Ann. ACM-SIAM Symp. on Discrete Algs.*, pp. 321–327. SIAM, San Fransico, CA, 1994.

[3] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rósen. Competitive non-preemptive call control. In *Proc. of the 5th Ann. ACM-SIAM Symp. on Discrete Algs.*, pp. 312–320. SIAM, San Fransico, CA, 1994.

[4] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proc. of the 35th Ann. Symp. on Foundations of Comp. Sci.*, pp. 412–423. IEEE, Computer Society Press, Los Alamitos, CA, 1994.

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

[6] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *Communications of the ACM*, 39(4):745–763, 1992.

[7] A. Feldmann, B. Maggs, J. Sgall, D. Sleator, and A. Tomkins. Competitive analysis of call admission algorithms that allow delay. Technical Report CMU-CS-95-102, School of Computer Science, CMU, Pittsburgh, PA 15213, 1995.

[8] J. Garay and I. Gopal. Call preemption in communication networks. In *Proc. of INFOCOM '92*, pp. 1043–1050. IEEE, Computer Society Press, Los Alamitos, CA, 1992.

[9] J. Garay, I. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *Proc. of the 2nd Israel Symp. on Theory and Computing Systems*, pp. 285–293. IEEE, Computer Society Press, Los Alamitos, CA, 1993.

[10] M. Goldwasser. Patience is a Virtue: The Effect of Slack on Competitiveness for Admission Control. In *Proc. of the 10th Ann. ACM-SIAM Symp. on Discrete Algs.*, to appear. SIAM, Baltimore, MD, 1999.

[11] A. Karlin, M. Manasse, L. Rudolpoh, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[12] R. Lipton and A. Tomkins. Online interval scheduling. In *Proc. of the 5th Ann. ACM-SIAM Symp. on Discrete Algs.*, pp. 302–311. SIAM, San Fransico, CA, 1994.

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