**Preprint from Journal of Scheduling, 6(2):183–211, 2003.**
**Copyright ©2003, Kluwer Academic Publishers**

# Patience is a Virtue: The Effect of Slack on Competitiveness for Admission Control*

Michael H. Goldwasser†
Dept. of Computer Science
Loyola University Chicago
6525 N. Sheridan Rd.
Chicago, IL 60626
mhg@cs.luc.edu

**Abstract**

We consider the online competitiveness for scheduling a single resource non-preemptively in order to maximize its utilization. Our work examines this model when parameterizing an instance by a new value which we term the *patience*. This parameter measures each job's willingness to endure a delay before starting, relative to this same job's processing time. Specifically, the *slack* of a job is defined as the gap between its release time and the last possible time at which it may be started while still meeting its deadline. We say that a problem instance has *patience* $\kappa$, if each job with length $\|J\|$ has a slack of at least $\kappa \cdot \|J\|$.

Without any restrictions placed on the job characteristics, previous lower bounds show that no algorithm, deterministic or randomized, can guarantee a constant bound on the competitiveness of a resulting schedule. Previous researchers have analyzed a problem instance by parameterizing based on the ratio between the longest job's processing time and the shortest job's processing time. Our main contribution is to provide a fine-grained analysis of the problem when simultaneously parameterized by patience and the range of job lengths. We are able to give tight or almost tight bounds on the deterministic competitiveness for all parameter combinations.

If viewing the analysis of each parameter individually, our evidence suggests that parameterizing solely on patience provides a richer analysis then parameterizing solely on the ratio of the job lengths. For example, in the special case where all jobs have the same length, we generalize a previous bound of 2 for the deterministic competitiveness with arbitrary slacks, showing that the competitiveness for any $\kappa \geq 0$ is exactly $1 + \frac{1}{\lfloor \kappa \rfloor + 1}$. Without any bound on the job lengths, a simple greedy algorithm is $(2 + \frac{1}{\kappa})$-competitive for any $\kappa > 0$. More generally we will find that for any fixed ratio of job lengths, the competitiveness of the problem tends towards 1 as the patience is increased. The converse is not true, as for any fixed $\kappa > 0$ we find that the competitiveness is bounded away from 1, no matter what further restrictions are placed on the ratio of job lengths.

*Keywords:* Patience, slack, admission control, scheduling, online analysis

*A preliminary version of this paper appear in *Proceedings of the Tenth Annual Symposium on Discrete Algorithms*, Baltimore, Maryland, Jan. 1999, pp. 396–405.

†Part of this work was completed while at Princeton University supported by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center.

# 1 Introduction

Imagine that a company rents use of a single resource, charging customers a fixed amount per minute. Job requests arrive, with each request specifying the length of time for which the customer will use the resource as well as a deadline by which the job should be completed. In the terminology of real-time systems we consider this a *firm deadline* [14]. The company is not required to complete each job, however it will only be paid for those jobs that are completed on or before their deadline. Therefore, a scheduling algorithm provides admission control as it decides which jobs to run, as well as when to run them. The goal is to maximize the successful use of the resource and thus the revenue for the company. We require that a schedule be *non-preemptive*, in that once a customer's job begins running, the job cannot be interrupted or terminated. The problem is online in nature as we assume that an algorithm has no knowledge about the existence of any job until the time at which a request arrives. When a job arrives, we assume that the algorithm is told both the length and deadline of the job. We analyze the performance of an algorithm using *competitive analysis*, comparing the schedule produced by an algorithm to the optimal schedule produced by an algorithm with full knowledge of future jobs [13, 19].

A typical commercial application of this model might be found in the photo-finishing lab of a large store. Let us presume that a customer enters with a roll of film, the developing of which requires a relatively small, but continuous, use of a photo-finishing machine. The customer might name a deadline for the job completion, generally quite large in proportion to the actual amount of machine time which is needed. He leaves his film with the company essentially committing to future payment for a completed job. The lab has flexibility in deciding when to run jobs, however it must use the machine non-preemptively, as any interruption in processing once it begins might cause permanent ruin to the negatives. Finally, an unfortunate customer might return at the chosen deadline only to learn that the lab has not even started servicing the order. At this point, the customer might state a new request for completing the job, but he might also walk away with his film while certainly under no obligation to make payment.

We can model many other consumer-bases services in this way. Of course the study of this problem has traditionally been motivated on a more industrial scale, generally involving real-time systems or communication-based applications. The resource might represent a channel of communication and a job (or call) represent a request for an uninterrupted connection through the channel for transmission of text, sound, images or video. Again, a deadline might be set after which the connection would be of no utility, and the slack offered by such a deadline might be proportionally larger than the required duration of a connection. The non-preemptive restriction models a form of customer relations. A provider may choose not to satisfy a request, however it may not begin servicing a request only to terminate it because a better customer comes along.

The *slack* of a job[1] is defined as 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 request of the most impatient of customers would have zero slack, meaning that this job must be given the resource immediately or else rejected. On the other hand, it may be that customers are reasonably more patient when forming requests. Our paper focuses on the effect of slack on the competitiveness of scheduling algorithms in this model. Specifically we say that a problem instance has *patience* $\kappa$ if each job $J$ of length $\|J\|$ offers a slack of at least $\kappa\|J\|$. If the patience of an instance is 0.05, for example, then a customer requesting 5 minutes of a resource must be willing to wait at least 15 seconds before beginning, whereas a customer requesting 5 hours of time must be willing to wait at least 15 minutes instead.

[1] This property has also been referred to as *delay*, *wait time* or *laxity* by previous researchers.

Measuring the slack in relation to a job's processing time seems quite natural for admission control. Yet with few exceptions, there has been relatively little work using this measure formally in analysis. The main contribution of our paper is to provide a fine-grained analysis of the problem taking into consideration the patience of an instance. We defer until Section 4 a more complete statement of our exact results. This will allow us to more formally define the precise model and notation in Section 2. In this light, Section 3 contains a review of related work previously done by researchers from the scheduling community. Our technical details are contained in Sections 5–8, and we conclude in Section 9.

# 2 Notation

We consider a job $J_j$ to be defined by a triple of non-negative values $\langle r_j, p_j, d_j \rangle$, where $r_j$ is the release time or arrival time of the job, $p_j$ is the length of the processing time, $d_j$ is the deadline for successful completion of the job. We denote as $x_j = d_j - p_j$, the *expiration time* of job $J_j$, namely the latest time at which the job can be started while still meeting its deadline. We denote as $s_j = x_j - r_j$, the slack of job $J_j$. W say that a job $J_j$ is *available* at time $t$ with respect to a (partial) schedule $\sigma$ if $r_j \leq t \leq x_j$ and if job $J_j$ has not been started in $\sigma$ prior to time $t$. Given a particular instance, we define the following two parameters. We let $\kappa = \min_j(s_j/p_j)$ denote the *patience* and we let $\Delta$ denote the ratio of the largest to smallest processing times. As needed in the analysis, we let $\{\Delta\} = \Delta - \lfloor \Delta \rfloor$ denote the fractional part of $\Delta$, and likewise $\{\kappa\} = \kappa - \lfloor \kappa \rfloor$. Our model will assume that the values of $\kappa$ and $\Delta$ are known to the scheduler a priori (though most of our algorithms do not use this information). For convenience, we assume that instances are normalized so that all processing times lie in the range $[1, \Delta]$.

We will use the notation $\|J_j\|$ to denote the processing time $p_j$ of a job, and for a set $\mathcal{S}$ of jobs we let $\|\mathcal{S}\|$ denote $\sum_{J \in \mathcal{S}} \|J\|$, the sum of the processing times of all jobs in $\mathcal{S}$. Given a schedule $\sigma$, we define the *gain* of the schedule to equal $\|\sigma\|$, and our goal is to maximize the gain. Following the standard notation of Graham *et. al.* [11, 15], our general problem is exactly an online version of $1 \mid r_j \mid \sum p_j \overline{U}_j$.

## 2.1 Competitive Analysis

We will measure the performance of an online algorithm by comparing the gain of the algorithm to the gain of the optimal offline algorithm, *opt*, that has full knowledge of the future when creating a schedule [13, 19]. We say that a (deterministic) online algorithm $A$ is *c-competitive* if $\text{gain}_{\text{opt}}(\mathcal{I}) \leq c \cdot \text{gain}_A(\mathcal{I})$, for all input instances $\mathcal{I}$. Finally, we define the *competitiveness of the problem* to be the competitiveness of the best possible (deterministic) online algorithm.

In order to characterize the competitiveness of this particular scheduling problem based on different values of $\kappa$ and $\Delta$, we introduce the following notation. We let $D(\kappa, \Delta)$ denote the deterministic competitiveness of the problem for a fixed $\kappa \geq 0$ and $\Delta > 1$. As was done by previous researchers, we study the special cases where all jobs have unit length, or where all jobs have one of two distinct lengths. We define $D_1(\kappa)$ to be the deterministic competitiveness for the case where all jobs must have equal length, and we define $D_2(\kappa, \Delta)$ to be the competitiveness for the case where all jobs have one of two distinct lengths, 1 or $\Delta$. We note that we could continue in defining terms $D_i(\kappa, \Delta)$ for values of $i > 2$, for cases where at most $i$ distinct job lengths exist. Our results, however, show that this hierarchy collapses in most cases, with $D_i(\kappa, \Delta) = D(\kappa, \Delta)$ for $i > 2$.

Although all but one of our results involve deterministic bounds, we also consider the performance of randomized online algorithms. In this case, the competitiveness compares 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 [7, 18]. We let $R(\kappa, \Delta)$, $R_1(\kappa)$, and $R_2(\kappa, \Delta)$ denote the *randomized* competitiveness for the corresponding settings.

## 3 Previous Work

The problem of online, non-preemptive scheduling of a single resource to maximize resource utilization was studied by Lipton and Tomkins in the case when *all jobs* have zero slack [16]. Note that this is a far more restrictive setting than the case where the patience of an instance is zero. This problem, termed *online interval scheduling*, was studied in a model where the value of $\Delta$ is unknown to the scheduler. 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 $\Delta$ is known results in bounds $R_2(0, \Delta) \geq 2 - \frac{1}{\Delta}$ and $R(0, \Delta)$ is $\Omega(\log \Delta)$.

Goldman, Parwatikar and Suri extended the model of Lipton and Tomkins, allowing jobs to specify arbitrary slacks [10]. They make no assumptions on the slacks specified by jobs and so this setting corresponds exactly to the case $\kappa = 0$ in our model. The existence of slack acts as a double-edged sword for competitive analysis. Clearly the added flexibility can only serve to increase the resource utilization for an algorithm. However maintaining competitiveness may become more difficult, as the optimal offline algorithm may know how to take better advantage of the flexibility than an online algorithm. For arbitrary job lengths, they give a randomized algorithm that proves $R(0, \Delta) \leq 6(\lceil \lg_2 \Delta \rceil + 1)$, matching the asymptotic lower bound of Lipton and Tomkins. When two job lengths are allowed, another randomized algorithm provides the bound $R_2(0, \Delta) \leq 4$. Additionally, they consider the case where all job lengths are identical, showing that a deterministic greedy algorithm is 2-competitive and that this is the best possible result, and thus $D(0) = 2$. They give a lower bound of $R(0) \geq \frac{4}{3}$, however without a matching upper bound. Additionally they proved a curious result, namely that if all slacks are equal to 1 or greater then the same greedy algorithm becomes $\frac{3}{2}$-competitive; thus $D(1) \leq \frac{3}{2}$. It is this result that is really the springboard of our own work. Our work generalizes this result for all values of $\kappa$ and re-examines the cases where job lengths vary. Goldman *et. al.* also consider a model in which they require that all jobs of the same length have the same slack, although the value of this slack need not have any relationship to the actual job length. With this additional requirement which they call *uniform delays*, they are able to improve upon some of their results because there is no longer an issue of deciding how to choose between two jobs of the same length. Although this model of uniform delays does not readily translate to our model, as $\kappa$ increases our results are significantly stronger than the bounds given for uniform delays.

Previous researchers have studied several closely related problems. Baruah *et. al.* study the *preemptive* setting of our problem for $\kappa = 0$, giving a 4-competitive algorithm for maximizing the use of a single processor with arbitrary job lengths [5]. They also show that this is the best possible (deterministic) algorithm for this setting. Scheduling a single resource is a special case of the general problem of call control in communication networks, where both admission 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] settings, with a more complete survey given by Plotkin [17].

Finally, the effect of requiring slack times proportional to processing times has been studied in several other models. The advantage of patience on competitiveness for scheduling single processors *preemptively* has been studied. Baruah and Haritsa give almost tight bounds for maximizing the

| Equal length jobs | | | | |
|---|---|---|---|---|
| | $D_1(\kappa)$ | | $R_1(\kappa)$ | |
| | LB | UB | LB | UB |
| $\kappa = 0$ | 2 | | 4/3 | |
| | From [10] | From [10] | From [10] | |
| $\kappa \geq 0$ | $1 + \frac{1}{\lfloor \kappa \rfloor + 1}$ | | $1 + \frac{1}{2\lfloor \kappa \rfloor + 3}$ | |
| | (Thm. 19) | (Thm. 11) | (Thm. 19) | |

Table 1: The lower/upper bounds for the deterministic/randomized competitiveness of the problem when all job lengths are equal. The best currently known randomized upper bounds are equivalent to the deterministic results.

utilization of a single processor [6], and Kalyanasundaram and Pruhs consider more general benefits associated with each job's completion [12]. The advantage of increased slack on competitiveness is also shown by Bar-Noy *et. al.* for a different problem specifically modeling movies-on-demand [4]. Their model assumes equal job lengths, as well as equal slack times, however in a setting where multiple requests can be serviced by a single channel if the requests are for the same movie. Finally, in a non-preemptive setting, Feldman *et. al.* study the requirement that slack time be proportional to processing time in the context of call control on networks [8]. In contrast to our results, they show that this additional requirement cannot be used to asymptotically improve the competitiveness of algorithms on networks of size $n$. They generalize previous lower bounds to include proportional slack, even when a network is a linear array and randomization is allowed. In light of this, our results demonstrate that this lack of improvement is inherently due to the routing aspects of the general problem and not due to the non-preemptive admission control.

# 4 Our Results

Our first set of results deals with a special case in which all jobs are required to have the same length (e.g. packets at a switch in an ATM network). We generalize the results of Goldman *et. al.* by providing tight deterministic bounds for all values of $\kappa$ and by improving on the randomized lower bounds for $\kappa > 0$. The lower bound is valid even if an algorithm is told the value of $\kappa$ in advance, whereas the upper bound is achieved by the same greedy algorithm as in Goldman *et. al.*, which does not need any knowledge of the parameter $\kappa$. These results are summarized in Table 1.

Our next set of results, summarized in Table 2, involves the case where all jobs have one of two distinct lengths. We are able to give matching upper and lower bounds for the deterministic competitiveness $D_2(\kappa, \Delta)$ for most combinations of $\kappa$ and $\Delta$. Again, the lower bounds are valid even if an algorithm is told the value of $\kappa$ and $\Delta$ in advance. The upper bounds are achieved by one of two different algorithms, neither of which uses explicit knowledge of the values of either $\kappa$ or $\Delta$. However, knowing which of the two algorithms to apply in achieving the best possible competitiveness depends on prior knowledge of the combination of parameter values.

We next consider the problem when instances may contain more than two distinct job lengths. What we find is that such a hierarchy essentially collapses as soon as three or more job lengths are allowed. We give a series of results, summarized in Table 3, proving tight or almost tight bounds on

| Two distinct job lengths | | |
|---|---|---|
| | $D_2(\kappa, \Delta)$ | |
| | LB | UB |
| $0 \le \kappa < \frac{1}{\Delta}$<br>$\Delta \ge \frac{1+\sqrt{5}}{2}$ | $1 + \Delta$<br>(Thm. 20) | (Cor. 7) |
| $0 \le \kappa < \frac{1}{\Delta}$<br>$\Delta \le \frac{1+\sqrt{5}}{2}$ | $2 + \frac{1}{\Delta}$<br>(Thm. 21) | (Cor. 7) |
| $\frac{1}{\Delta} \le \kappa < 1$ | $2 + \frac{\lceil \Delta \rceil - 1}{\Delta}$<br>(Thm. 21) | (Thm. 17) |
| $1 \le \kappa < \Delta$ | $1 + \frac{\lceil \Delta \rceil}{\Delta}$<br>(Thm. 22) | (Thm. 18) |
| $\Delta \le \kappa \le 1 + \frac{1}{\Delta} \le 2$ | $1 + \frac{2}{\Delta+1}$<br>(Thm. 23) | $2 + \frac{1}{\kappa} - \frac{1}{\Delta+1}$<br>(Thm. 14) |
| $\Delta \le \min(2, \kappa)$<br>$1 + \frac{1}{\Delta} \le \kappa$ | $1 + \frac{2}{\Delta + \lfloor \kappa - \Delta \rfloor + 1}$<br>(Thm. 23) | $1 + \frac{2}{\kappa}$<br>(Cor. 13) |
| $2 < \Delta \le \kappa$ | $1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1}$<br>(Thm. 23) | (Cor. 13) |

Table 2: Bounds for the deterministic competitiveness, parameterized simultaneously by $\kappa$ and $\Delta$, when jobs have one of two distinct lengths.

| Arbitrary job lengths | | |
|---|---|---|
| | $D(\kappa, \Delta)$ | |
| | LB | UB |
| $0 \leq \kappa < \frac{1}{\Delta}$ | $2 + \Delta$ (Thm. 24) | (Thm. 4) |
| $\frac{1}{\Delta} \leq \kappa \leq \lceil \Delta \rceil - 1$ | $2 + \frac{1}{\kappa}$ (Thm. 25) | (Thm. 5) |
| $\lceil \Delta \rceil - 1 < \kappa < \Delta$ | $2 + \frac{1}{\kappa} - \frac{\{\kappa\}}{\kappa}$ (Cor. 27) | $2 + \frac{1}{\kappa}$ (Thm. 5) |
| $\Delta \leq \kappa < \Delta + 1$ $0<\{\Delta\}\leq\{\kappa\}$ | 2 (Cor. 27) | $2 + \frac{1}{\kappa} - \frac{1}{\Delta+1}$ (Thm. 14) |
| $\Delta \leq \kappa < \Delta + 1$ $\{\Delta\}=0$ or $\{\Delta\}>\{\kappa\}$ | $2 - \frac{\{\kappa\}}{\kappa}$ (Cor. 27) | $2 + \frac{1}{\kappa} - \frac{1}{\Delta+1}$ (Thm. 14) |
| $\Delta + 1 \leq \kappa$ $0<\{\Delta\}\leq\{\kappa\}$ | $1 + \frac{\lceil \Delta \rceil}{\lfloor \kappa \rfloor + 1}$ (Thm. 26) | $1 + \frac{\Delta+1}{\kappa}$ (Thm. 14) |
| $\Delta + 1 \leq \kappa$ $\{\Delta\}=0$ or $\{\Delta\}>\{\kappa\}$ | $1 + \frac{\lceil \Delta \rceil}{\kappa}$ (Thm. 26) | $1 + \frac{\Delta+1}{\kappa}$ (Thm. 14) |

Table 3: Bounds for the deterministic competitiveness when parameterized both by $\kappa$ and $\Delta$. Lower bounds apply with three distinct job lengths; upper bounds apply with arbitrarily many job lengths.

the deterministic competitiveness with three or more distinct job lengths. Our results are robust in the following ways. The lower bounds again assume that an algorithm has full knowledge of both parameter values. More so, all of the lower bound are proven using constructions which involve at most three distinct job lengths. Conversely, all of our upper bounds apply with arbitrarily many distinct job lengths, and are proven with a simple greedy algorithm that uses no explicit knowledge of the values of either $\kappa$ or $\Delta$.

Finally, we attempt to evaluate the relative influence of the two separate parameters by considering various cross-sections of our results. To this end, we consider the supremum and infimum of the competitiveness bounds when one parameter is fixed. Our evidence suggests that parameterizing solely on patience provides a richer analysis then parameterizing solely on the ratio of the job lengths. In particular, we find that for any fixed ratio of job lengths, the competitiveness of each problem tends towards 1 as the patience is increased. Conversely, for any fixed $\kappa > 0$ we find that the competitiveness is bounded strictly away from 1, no matter what further restrictions are placed on the ratio of job lengths. Table 4 provides the precise limits, with discussion deferred to Section 7.

| | $\inf_{\kappa}$ | $\sup_{\kappa}$ | $\inf_{\Delta}$ | $\sup_{\Delta}$ |
|---|---|---|---|---|
| $D_2(\kappa, \Delta)$ | 1 | $\max(1+\Delta, 2+\frac{1}{\Delta})$ | $\geq \min\left(2, 1+\frac{2}{\kappa+1}\right)$ | $1+\max\left(\frac{\lceil\kappa\rceil+1}{\lceil\kappa\rceil}, \frac{\lfloor\kappa\rfloor+1}{\kappa}\right)$ |
| $D(\kappa, \Delta)$ | 1 | $2+\Delta$ | $\geq \left(1+\frac{2}{\lfloor\kappa\rfloor+1}\right)$ | $2+\frac{1}{\kappa}$ |

Table 4: Deterministic competitiveness of the problems when only one parameter is fixed.

# 5 Deterministic Online Algorithms

We begin by considering a general class of online algorithms.

**Definition 1** *A greedy-type algorithm is one which satisfies the following two conditions:*

1. *The algorithm never allows the processor to idle when an available job is in the system.*
2. *Whenever the processor is idle, all previous state information is lost.*

This second condition equivalently states that when the system becomes void of available jobs, its future behavior is independent of the past. This is typical of real-time systems. Many different greedy-type algorithms exist, as a choice remains as to which job to run at any point when several jobs are available. Typical heuristics include scheduling the available job with the longest processing time, the shortest processing time, the earliest deadline, the earliest expiration, or the least laxity. In some cases, we can show that any such greedy-type algorithm achieves the strongest possible deterministic competitiveness.

In a majority of other cases, we will need to examine specific greedy-type algorithms in order to achieve the strongest competitiveness. In particular we denote GREEDY to be the algorithm which always runs the available job with the *earliest expiration time*. When all jobs have equal length, a simple transposition argument shows that on any instance, this is the most profitable of all greedy-type algorithms. With differing job lengths, GREEDY is not the best choice on all instances, however for a majority of settings we are able to show that the GREEDY algorithm indeed achieves the strongest possible deterministic competitiveness. In some special cases involving instances with exactly two distinct job lengths, we show that the strongest deterministic competitiveness is achieved by a related algorithm, GREEDY-TWOLENGTHS, which favors large jobs over small jobs when scheduling.

## 5.1 Competitiveness of greedy-type algorithms

We let $\sigma$ denote the schedule produced by a particular greedy-type algorithm $A$ when run on an instance $\mathcal{I}$. We define the *busy periods of* $\sigma$ as the maximal intervals during which the resource is continuously in use. The first property of a greedy-type algorithm assures us that when a busy period ends, no jobs are available. The processor remains idle until the next job arrival. The second property of a greedy-type algorithm assures us that from this point onward, the behavior of the algorithm is as if we are restarting the algorithm on a new instance. For this reason, we are able to show that without loss of generality, we can analyze such greedy-type algorithms on instances which result in a single busy period.

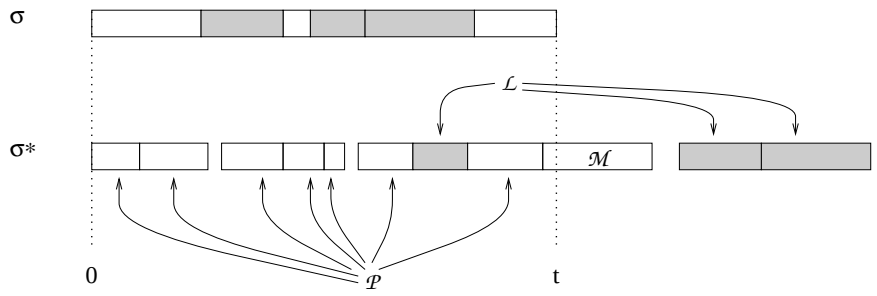


Figure 1: Example of the analysis of a busy period. Jobs of $\mathcal{L}$ are shaded.

**Lemma 2** *If a deterministic greedy-type algorithm $A$ is $c$-competitive when run on instances which result in a single busy period, than $A$ is $c$-competitive on all instances. The same statement is true when further restricting instances with bounds on $\Delta$, $\kappa$ and the number of distinct job lengths.*

*Proof:* We prove this claim by induction on the number of jobs. In general, we will assume the hypothesis for instances with $n$ or fewer jobs, and prove it for an arbitrary instance $\mathcal{I}$ with $n+1$ jobs. If algorithm $A$ produces more than one busy period on $\mathcal{I}$, then we let $t'$ be the latest time at which $A$ starts a job immediately after an idle period. We define two new instances $\mathcal{I}_1$ and $\mathcal{I}_2$, where $\mathcal{I}_1$ contains all jobs of $\mathcal{I}$ which arrive strictly before $t'$ and where $\mathcal{I}_2$ contains all jobs of $\mathcal{I}$ which arrive on or after time $t'$. The properties of a deterministic greedy-type algorithm are defined precisely to assure us that $\text{gain}_A(\mathcal{I}) = \text{gain}_A(\mathcal{I}_1) + \text{gain}_A(\mathcal{I}_2)$. We do not necessarily have the same equality for the optimal schedule. It may be that jobs of $\mathcal{I}_1$ exist with deadlines which are greater than the arrival times of jobs in $\mathcal{I}_2$. Thus it is not necessarily possible to overlay the optimal schedules for $\mathcal{I}_1$ and $\mathcal{I}_2$ without collisions. It is clear, however, that $\text{gain}_{\text{opt}}(\mathcal{I}) \leq \text{gain}_{\text{opt}}(\mathcal{I}_1) + \text{gain}_{\text{opt}}(\mathcal{I}_2)$, as we may always partition the optimal schedule for $\mathcal{I}$ into *feasible* schedules for $\mathcal{I}_1$ and $\mathcal{I}_2$. Since both $\mathcal{I}_1$ and $\mathcal{I}_2$ have at most $n$ jobs, we can apply induction to assume the $c$-competitiveness of $A$, namely that $\text{gain}_{\text{opt}}(\mathcal{I}_1) \leq c \cdot \text{gain}_A(\mathcal{I}_1)$, and similarly for $\mathcal{I}_2$. Therefore,

$$\text{gain}_{\text{opt}}(\mathcal{I}) \leq \text{gain}_{\text{opt}}(\mathcal{I}_1) + \text{gain}_{\text{opt}}(\mathcal{I}_2) \leq c \cdot \text{gain}_A(\mathcal{I}_1) + c \cdot \text{gain}_A(\mathcal{I}_2) = c \cdot \text{gain}_A(\mathcal{I}),$$

proving the competitiveness of $A$ on $\mathcal{I}$. ∎

We now introduce some additional notation for analyzing the competitiveness of greedy-type algorithms on a single busy period. Without loss of generality, we assume that the first job arrives at time 0 and that the schedule $\sigma$ produced by a greedy-type algorithm occupies the resource during the interval $[0, t)$. We let $\sigma^*$ denote the optimal schedule on the instance and we partition the jobs scheduled in $\sigma^*$ into three distinct groups:

- $\mathcal{P} \subseteq \sigma^*$ are those jobs with expiration time strictly less than $t$ and which are completed in $\sigma^*$ strictly before time $t$ ("prompt" jobs).
- $M \in \sigma^*$ is a job, if one exists which has expiration time strictly less than $t$, but which is *completed* in $\sigma^*$ on or after time $t$ ("middle" job).
- $\mathcal{L} \subseteq \sigma^*$ contains those jobs which have an expiration time of $t$ or greater ("late" jobs).

We begin with some initial observations. The general analysis framework is pictured in Figure 1.

**Lemma 3** *For any greedy-type algorithm and any instance comprising a single busy period,*
*(1).* $\|\sigma^*\| = \|\mathcal{P}\| + \|M\| + \|\mathcal{L}\|$, *(2).* $\|\mathcal{P}\| < \|\sigma\|$, *(3).* $\mathcal{L} \subseteq \sigma$.

*Proof:* (1) and (2) follow by definition. (3) Each job of $\mathcal{L}$ has an expiration time of $t$ or greater. At the same time, $\sigma$ leaves the processor idle from time $t$ onward. This would not happen if any job of $\mathcal{L}$ was available to $\sigma$ at time $t$ or later, thus all such jobs must have been scheduled in $\sigma$. ∎

**Theorem 4** *For $\kappa \geq 0$ and all $\Delta$,*

$$D(\kappa, \Delta) \leq 2 + \Delta,$$

*with the upper bound achieved by any greedy-type algorithm.*

*Proof:* We analyze the competitiveness of a greedy-type algorithm on a single busy period, as per Lemma 2. We utilize the facts that $\|\mathcal{P}\| < \|\sigma\|$, $\|\mathcal{L}\| \leq \|\sigma\|$, $\|M\| \leq \Delta$, and $\|\sigma\| \geq 1$ to conclude, $\|\sigma^*\| = \|\mathcal{P}\| + \|M\| + \|\mathcal{L}\| < \|\sigma\| + \Delta + \|\sigma\| = 2\|\sigma\| + \Delta \leq (2 + \Delta)\|\sigma\|$. ∎

**Theorem 5** *For $\kappa > 0$ and all $\Delta$,*

$$D(\kappa, \Delta) \leq 2 + \frac{1}{\kappa},$$

*with the upper bound achieved by any greedy-type algorithm.*

*Proof:* We analyze the competitiveness of a greedy-type algorithm on an instance consisting of a single busy period, as per Lemma 2. If $M \in \sigma$ or if $M$ does not exist, then $\{M\} \cup \mathcal{L} \subseteq \sigma$ and $\|\sigma^*\| = \|\mathcal{P}\| + \|M\| + \|\mathcal{L}\| < \|\sigma\| + \|\sigma\| = 2\|\sigma\|$. Alternatively if $M$ exists but $M \notin \sigma$, we consider the patience requirement which states that $M$ has a slack of at least $\kappa\|M\|$. A greedy-type algorithm could never have left the processor idle at a point when $M$ was available and therefore $\|\sigma\| \geq \kappa\|M\|$. Combining this with the bounds that $\|\mathcal{P}\| < \|\sigma\|$ and $\|\mathcal{L}\| \leq \|\sigma\|$, we see $\|\sigma^*\| = \|\mathcal{P}\| + \|M\| + \|\mathcal{L}\| \leq \|\sigma\| + \frac{1}{\kappa}\|\sigma\| + \|\sigma\| = (2 + \frac{1}{\kappa})\|\sigma\|$. ∎

**Theorem 6** *For all $\kappa$ and $\Delta$,*

$$D_2(\kappa, \Delta) \leq \max(1 + \Delta, 2 + \frac{1}{\Delta}),$$

*with the upper bound achieved by any greedy-type algorithm.*

*Proof:* We analyze the competitiveness of a greedy-type algorithm on a single busy period, as per Lemma 2. Using Lemma 3, we know that $\|\sigma^*\| = \|\mathcal{P}\| + \|M\| + \|\mathcal{L}\| \leq \|\mathcal{P}\| + \|M\| + \|\sigma\|$. What remains is to show that $\|\mathcal{P} \cup \{M\}\| \leq d\|\sigma\|$, where $d = \max(\Delta, \frac{\Delta+1}{\Delta})$, completing the proof.

We label the jobs of $\sigma$ as $J_0, J_1, \ldots, J_n$. We know that each job of $\mathcal{P} \cup \{M\}$ must begin strictly before time $t$ in $\sigma^*$. Therefore each job of $\mathcal{P} \cup \{M\}$ began in $\sigma^*$ at a time when some job $J_i$ was running in $\sigma$. We partition the jobs of $\mathcal{P} \cup \{M\}$ into groups $G_1, G_2, \ldots, G_n$ such that jobs in group $G_i$ start in $\sigma^*$ while job $J_i$ is running in $\sigma$. We will show that $\|G_i\| \leq d\|J_i\|$ for all $i$, and thus that $\|\mathcal{P} \cup \{M\}\| = \sum \|G_i\| \leq \sum d\|J_i\| = d\|\sigma\|$.

If $J_i$ is a unit-length job, then we claim that $G_i$ contains at most one job, as any job which starts in $\sigma^*$ while $J_i$ is running will complete on or after $J_i$ completes. In this case $\|G_i\| \leq \Delta \leq d \cdot 1 = d\|J_i\|$.

Alternatively $J_i$ is a job of length $\Delta$. In this case we point out that $G_i$ contains at most one large job. Also, at most $\lceil\Delta\rceil - 1$ small jobs may be started along with one large job. Therefore $\|G_i\| \leq \lceil\Delta\rceil - 1 + \Delta$. If $\Delta < 2$, then we know that$\lceil\Delta\rceil = 2$ and thus $\|G_i\| \leq 1 + \Delta \leq d\Delta = d\|J_i\|$. If $\Delta \geq 2$, then we have that $\|G_i\| \leq 2\Delta \leq \Delta \cdot \Delta \leq d\Delta = d\|J_i\|$. ∎

**Corollary 7** *For* $1 \leq \Delta \leq \frac{1+\sqrt{5}}{2}$,

$$D_2(\kappa, \Delta) \leq 2 + \frac{1}{\Delta}\,.$$

*For* $\Delta \geq \frac{1+\sqrt{5}}{2}$,

$$D_2(\kappa, \Delta) \leq 1 + \Delta\,.$$

*Proof:* The equation $1 + \Delta = 2 + \frac{1}{\Delta}$ is solved by $\phi = \frac{1+\sqrt{5}}{2}$, the golden ratio. When $\Delta \leq \phi$, the expression from Theorem 6 is maximized by the term $2 + \frac{1}{\Delta}$, and when $\Delta \geq \phi$, the expression is maximized by the term $1 + \Delta$. ∎

### 5.2 Stronger bounds for GREEDY

Recall that we have denoted as GREEDY the greedy-type algorithm which always schedules the available job with the earliest expiration time. From Lemma 2, we know that it suffices to analyze GREEDY on individual busy periods. It turns out that we can examine a significantly more narrow subset of instances when analyzing GREEDY. We define a *fundamental instance* $\mathcal{I}$ to be one which satisfies the following conditions:

1. GREEDY utilizes the resource continuously from time 0 up to some time $t$.
2. If we let $J$ denote the job which GREEDY starts at time 0, each job of $\mathcal{I} - J$ has an expiration time which is strictly less than $t$.

**Lemma 8** *If the* GREEDY *algorithm is $c$-competitive when run on fundamental instances, then it is $c$-competitive on all instances. The same statement is true when further restricting instances with a bound on $\Delta$, $\kappa$ or the number of distinct job lengths.*

*Proof:* We know from Lemma 2 that it suffices to prove the competitiveness of GREEDY on instances which result in a single busy period. With this in mind, we develop an inductive hypothesis stating that if GREEDY is $c$-competitive when run on fundamental instances, then it is $c$-competitive on all instances with $n$ or fewer jobs which results in a single busy period.

We assume the result for instances with $n$ or fewer jobs, and let $\mathcal{I}$ be a non-fundamental instance which consists of $n+1$ jobs and which results in a single busy period for GREEDY. Without loss of generality, we assume the period is $[0, t)$, and that job $J$ is started at time 0. We let $\mathcal{L}$ denote the set of jobs which have expiration time of $t$ or greater. If $\mathcal{L} = \{J\}$, then $\mathcal{I}$ is fundamental; otherwise we will show how to effectively divide this instance into two independent parts applying our induction. Lemma 3 assures that each job of $\mathcal{L}$ is completed by GREEDY. We denote as $0 < t' < t$ the latest time at which GREEDY starts a job of $\mathcal{L}$, and we let $J' \neq J$ denote that job which is started.

Rather than dividing instance $\mathcal{I}$, we define an altered instance $\mathcal{I}'$ by modifying job $J'$ to have release time $t'$ and infinite deadline. We claim that GREEDY produces the same schedule for $\mathcal{I}'$ as it did for $\mathcal{I}$ because $J'$ is the only job available to GREEDY at time $t'$. All other late jobs have already been completed by $t'$ and had a non-late job been available, it would have originally been chosen ahead of $J'$. Equally important, the optimal gain on instance $\mathcal{I}'$ is at least as large as the optimal gain on $\mathcal{I}$ due to the increased availability of $J'$.

Now, we partition instance $\mathcal{I}'$ into two instances $\mathcal{I}'_1$ and $\mathcal{I}'_2$, where $\mathcal{I}'_1$ consists of all jobs of $\mathcal{I}'$ which arrive strictly before time $t'$, and $\mathcal{I}'_2$ consists of those jobs which arrive on or after time $t'$. It is clear that $\text{gain}_{\text{GREEDY}}(\mathcal{I}') = \text{gain}_{\text{GREEDY}}(\mathcal{I}'_1) + \text{gain}_{\text{GREEDY}}(\mathcal{I}'_2)$. We also have that $\text{gain}_{\text{opt}}(\mathcal{I}') \leq \text{gain}_{\text{opt}}(\mathcal{I}'_1) + \text{gain}_{\text{opt}}(\mathcal{I}'_2)$ as we may always partition the optimal schedule for $\mathcal{I}'$ into feasible schedules for $\mathcal{I}'_1$ and $\mathcal{I}'_2$. Since $J \in \mathcal{I}'_1$ and $J' \in \mathcal{I}'_2$, both instances have at most $n$ jobs and our induction

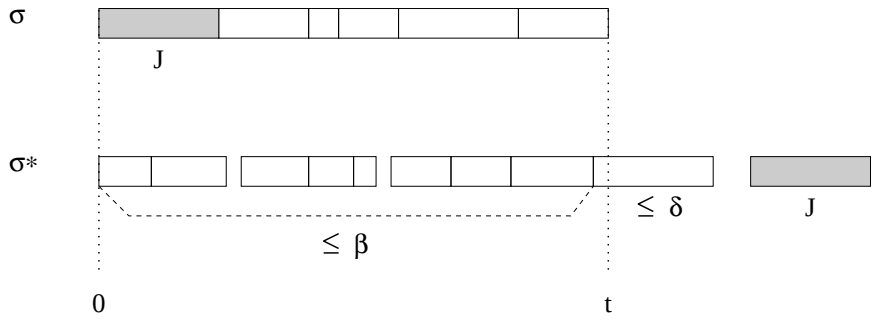


Figure 2: Analysis of a fundamental instance.

applies. Therefore, $\text{gain}_{\text{opt}}(\mathcal{I}'_1) \leq c \cdot \text{gain}_{\text{GREEDY}}(\mathcal{I}'_1)$ and $\text{gain}_{\text{opt}}(\mathcal{I}'_2) \leq c \cdot \text{gain}_{\text{GREEDY}}(\mathcal{I}'_2)$, and we conclude that

$$\begin{aligned}\text{gain}_{\text{opt}}(\mathcal{I}) &\leq \text{gain}_{\text{opt}}(\mathcal{I}') \leq \text{gain}_{\text{opt}}(\mathcal{I}'_1) + \text{gain}_{\text{opt}}(\mathcal{I}'_2) \\ &\leq c \cdot \text{gain}_{\text{GREEDY}}(\mathcal{I}'_1) + c \cdot \text{gain}_{\text{GREEDY}}(\mathcal{I}'_2) = c \cdot \text{gain}_{\text{GREEDY}}(\mathcal{I}') = c \cdot \text{gain}_{\text{GREEDY}}(\mathcal{I}).\end{aligned}$$

This proves the competitiveness of GREEDY on instance $\mathcal{I}$ and thus completes the induction. ∎

In analyzing the competitiveness of GREEDY on a fundamental instance $\mathcal{I}$, we use notation which is similar, yet significantly simpler, than that used in Section 5.1. We again assume that GREEDY schedules the resource continuously from time 0 to some time $t$. We let $J$ be the job which is run by GREEDY at time 0. If $J$ is the only job in the instance, than clearly GREEDY achieves optimality. Therefore, we assume that one or more other jobs exist and we define $\delta$ to be the size of the largest job in the set $\mathcal{I} - J$. Finally, we define the value $\beta$ to be the maximal achievable gain on jobs of $\mathcal{I} - J$ for a schedule completing strictly before time $t$.

**Lemma 9** *For a fundamental instance $\mathcal{I}$, the competitiveness of* GREEDY *can be upper bounded as*

$$\frac{\|\sigma^*\|}{\|\sigma\|} \leq \frac{\beta + \delta + \|J\|}{t}.$$

*Proof:* Consider the jobs achieved in the optimal schedule $\sigma^*$. By the definition of a fundamental instance, job $J$ is the only one which may be started on or after time $t$. One job of $\mathcal{I} - J$ may start strictly before time $t$ and complete on or after time $t$, but such a job has length at most $\delta$. Finally, all of the other jobs are completed during the range $[0, t - \epsilon]$ for some $\epsilon > 0$. By definition, they can contribute at most $\beta$ to the gain of the optimal schedule. Therefore, $\|\sigma^*\| \leq \beta + \delta + \|J\|$ . This analysis is pictured in Figure 2. ∎

**Lemma 10** *If a fundamental instance $\mathcal{I}$ contains more than one job, then $t > \kappa\delta$.*

*Proof:* By definition, there exists a job with length $\delta$ in $\mathcal{I} - J$. This job arrives on or after time 0 and due to the patience, it has an expiration time of $\kappa\delta$ or greater. At the same time, because the instance is fundamental, it must be that the job has an expiration time strictly less than $t$. Combining these inequalities, we have that $\kappa\delta < t$. ∎

**Theorem 11** *For all $\kappa$,*

$$D_1(\kappa) \leq \left(1 + \frac{1}{\lfloor \kappa \rfloor + 1}\right),$$

*with the upper bound achieved by* GREEDY.

*Proof:* Using Lemma 8, we consider an arbitrary fundamental instance $\mathcal{I}$. Since all jobs have unit length $\|J\| = 1$, $\delta = 1$ and $t$ must be integral. Furthermore, it must be that $\beta \leq t - 1$, since at most $t - 1$ unit-length jobs can be completed strictly before time $t$. Applying Lemma 9, the competitiveness of GREEDY on this instance is upper bounded by,

$$\frac{\|\sigma^*\|}{\|\sigma\|} \leq \frac{\beta + \delta + \|J\|}{t} \leq \frac{t+1}{t} = 1 + \frac{1}{t}\,.$$

Finally, Lemma 10 assures us that $t > \kappa$ and since $t$ is integral we have that $t \geq \lfloor \kappa \rfloor + 1$. Therefore the overall competitiveness,

$$\frac{\|\sigma^*\|}{\|\sigma\|} \leq 1 + \frac{1}{t} \leq 1 + \frac{1}{\lfloor \kappa \rfloor + 1}\,.$$

∎

**Theorem 12** *For all $\kappa \geq \Delta$,*

$$D_2(\kappa, \Delta) \leq \max\left(1 + \frac{2}{\kappa}, 1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1}\right),$$

*with the upper bound achieved by* GREEDY.

*Proof:* Using Lemma 8, we consider an arbitrary fundamental instance $\mathcal{I}$. We consider two possible cases, depending on whether $\delta = \Delta$ or $\delta = 1$. If $\delta = \Delta$, then Lemma 10 assures that $t > \kappa\Delta$. We can combine this inequality, together with the inequalities $\beta < t$ and $\|J\| \leq \Delta$, in applying Lemma 9. This bounds the competitiveness of GREEDY on this instance by,

$$\frac{\|\sigma^*\|}{\|\sigma\|} \leq \frac{\beta}{t} + \frac{\delta + \|J\|}{t} < 1 + \frac{\Delta + \Delta}{\kappa\Delta} = 1 + \frac{2}{\kappa}\,.$$

Alternatively, we consider the case where $\delta = 1$, and thus all jobs of $\mathcal{I} - J$ are unit-length. If $\|J\| = 1$, than all jobs are unit length, and we can rely on Theorem 11, which assures us that the competitiveness is upper bounded by

$$\frac{\|\sigma^*\|}{\|\sigma\|} \leq 1 + \frac{1}{\lfloor \kappa \rfloor + 1} \leq 1 + \frac{2}{\kappa}\,.$$

Otherwise, we have that $\delta = 1$ and $\|J\| = \Delta$. Since all other jobs are unit length, it must be the case that $\beta$ is integral, and that $t = \Delta + a$ for some non-negative integer $a$. Lemma 10, combined with the integrality of $\beta$ assures us that $\beta \leq \lceil t \rceil - 1 = \lceil \Delta \rceil + a - 1$. Now we apply Lemma 9, to show that the competitiveness is bounded by,

$$\frac{\|\sigma^*\|}{\|\sigma\|} \leq \frac{\beta + \delta + \|J\|}{t} \leq \frac{\lceil \Delta \rceil + a - 1 + 1 + \Delta}{\Delta + a} = 1 + \frac{\lceil \Delta \rceil}{\Delta + a}\,.$$

Finally, Lemma 10 tells us that $t = \Delta + a > \kappa$. Since $a$ is integral and $a > \kappa - \Delta$, we can infer that $a \geq \lfloor \kappa - \Delta \rfloor + 1$. Combining this fact with the previous bound on the competitiveness, we conclude

$$\frac{\|\sigma^*\|}{\|\sigma\|} \leq 1 + \frac{\lceil \Delta \rceil}{\Delta + a} \leq 1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1}\,.$$

∎

**Corollary 13** *For* $1 < \Delta \leq \min(2, \kappa)$,

$$D_2(\kappa, \Delta) \leq 1 + \frac{2}{\kappa},$$

*and for* $2 < \Delta \leq \kappa$,

$$D_2(\kappa, \Delta) \leq 1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1},$$

*with the upper bounds achieved by* GREEDY.

*Proof:* We note that the expression $\Delta + \lfloor \kappa - \Delta \rfloor + 1 = \kappa + 1 - ((\kappa - \Delta) - \lfloor \kappa - \Delta \rfloor)$, and therefore that $\kappa < \Delta + \lfloor \kappa - \Delta \rfloor + 1 \leq \kappa + 1$. When $1 < \Delta \leq 2$, we see that

$$1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1} \quad = \quad 1 + \frac{2}{\Delta + \lfloor \kappa - \Delta \rfloor + 1} < 1 + \frac{2}{\kappa}$$

and so the maximum from Theorem 12 is achieved by $1 + \frac{2}{\kappa}$. Alternatively, when $2 < \Delta \leq \kappa$,

$$1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1} \quad \geq \quad 1 + \frac{3}{\kappa + 1} = 1 + \frac{2+1}{\kappa + 1} > 1 + \frac{2}{\kappa}$$

as we know that $2 < \kappa$. Therefore in this case the maximum from Theorem 12 is achieved by the second expression. ∎

**Theorem 14** *For all* $\kappa \geq \Delta$,

$$D(\kappa, \Delta) \leq \min\left(2 + \frac{1}{\kappa} - \frac{1}{\Delta + 1}, 1 + \frac{\Delta + 1}{\kappa}\right),$$

*with the upper bound achieved by* GREEDY.

*Proof:* Using Lemma 8, we consider a non-trivial fundamental instance $\mathcal{I}$. Based on Lemma 10, $t > \kappa\delta \geq \Delta \geq \|J\|$. This assures us that the schedule produced by GREEDY cannot consist solely of job $J$, and therefore, it must be that $t \geq \|J\| + 1$ since the minimum additional job size is 1. Therefore, $t \geq \max(\kappa\delta, \|J\| + 1)$. Now, we combine the fact that $\beta < t$ with Lemma 9 to bound the competitiveness of GREEDY on this instance by, $1 + \frac{1}{\kappa} + \frac{\|J\|}{t}$. Finally we can bound the term,

$$\frac{\|J\|}{t} \leq \min\left(\frac{\|J\|}{\|J\| + 1}, \frac{\|J\|}{\kappa\delta}\right) = \min\left(1 - \frac{1}{\|J\| + 1}, \frac{\|J\|/\delta}{\kappa}\right) \leq \min\left(1 - \frac{1}{\Delta + 1}, \frac{\Delta}{\kappa}\right),$$

completing the proof. ∎

### 5.3 Some special cases involving two distinct lengths

Lower bounds in Section 6 will demonstrate that GREEDY achieves the best possible deterministic competitiveness for the majority of settings. There are, however, a few situations in which the GREEDY algorithm does not achieve the best possible competitiveness. In particular, when instances involve jobs of two distinct lengths, there are times when we must consider a different algorithm to achieve the strongest competitiveness. Throughout this section, we refer to jobs of length 1 as "small" jobs, and jobs of length $\Delta$ as "large" jobs.

A simple example can be constructed for which GREEDY performs poorly. Let instance $\mathcal{I}$ consist of three jobs $J_1 = \langle 0, 1, \infty \rangle$, $J_2 = \langle \epsilon, \Delta, \Delta + 1 + \epsilon \rangle$, and $J_3 = \langle \epsilon, 1, 2 \rangle$ for arbitrarily small $\epsilon > 0$. The

patience of this instance is equal to $\kappa = \frac{1}{\Delta}$ due to job $J_2$. The optimal schedule for this instance is to start job $J_3$ at time $\epsilon$, job $J_2$ at time $1+\epsilon$, and job $J_1$ after that. Unfortunately, GREEDY will run job $J_1$ at time 0, job $J_3$ at time 1, while job $J_2$ will be lost. Therefore, on this instance

$$\frac{\text{gain}_{\text{opt}}(\mathcal{I})}{\text{gain}_{\text{GREEDY}}(\mathcal{I})} = \frac{2+\Delta}{2} = 1 + \frac{\Delta}{2}\,.$$

It is possible to do better in this situation. Specifically, we will see in Theorem 17 that it is possible to guarantee a competitiveness which is strictly less than 3 in the case where $\kappa = \frac{1}{\Delta}$ and all jobs have one of two known lengths. The algorithm which we will use in this section is a greedy-type algorithm which gives preference to larger jobs.

**Definition 15** *We define the algorithm* GREEDY-TWOLENGTHS *as follows. If there exists an available large job when the resource is free, the algorithm schedules the available large job with the earliest expiration time. If no large jobs are available but one or more small jobs are available, then the algorithm schedules the available small job with the earliest expiration time.*

By Lemma 2, we can analyze the performance of GREEDY-TWOLENGTHS on an instance $\mathcal{I}$ which comprises a single busy period. We let $\sigma$ denote the schedule produced by the algorithm, and we assume the resource is used during the interval $[0, t)$. With GREEDY we were able to group jobs into fundamental instances for further analysis. Because of the interplay between scheduling large jobs and small jobs in GREEDY-TWOLENGTHS, we will take a slightly different approach to grouping jobs for analysis.

In order to partition the large jobs of $\mathcal{I}$, we will define what we term blocks. We will call a set of large jobs $B_i \subseteq \mathcal{I}$ a *regular block* if there exist some interval $[s_i, t_i)$ such that:

- Throughout $[s_i, t_i)$, only jobs from $B_i$ are processed in $\sigma$.
- With the possible exception of the job which $\sigma$ begins at time $s_i$, all jobs of $B_i$ arrive on or after time $s_i$ and expire strictly before time $t_i$.

We will call a set of large jobs $B_i \subseteq \mathcal{I}$ a *delayed block* if if there exist some interval $[s_i, t_i)$ such that:

- A small job is run in $\sigma$ during $[s_i - 1, s_i)$.
- Throughout $[s_i, t_i)$, only jobs from $B_i$ are processed in $\sigma$.
- All jobs of $B_i$ arrive strictly after time $s_i - 1$ and expire strictly before time $t_i$.

**Lemma 16** *If $\kappa \geq \frac{1}{\Delta}$, we can partition all large jobs into blocks, given an instance comprising a single busy period.*

*Proof:* We will begin by defining sets containing all of those large jobs which successfully ran in $\sigma$. Large jobs which did not run in $\sigma$ will later be assigned to the existing sets.

Consider the last large job to be completed in $\sigma$. We assign this job into a new set $B_i$ and we let $t_i$ be the time at which this job was completed in $\sigma$. If this large job has an expiration time strictly less than $t_i$ and if it was preceded by a large job in $\sigma$, then we assign that preceding large job into this same set $B_i$. We continue assigning jobs into $B_i$ until we either reach a job with expiration time of $t_i$ or greater, or until we reach a large job which was not preceded by another large job in $\sigma$. This defines time $s_i$ associated with the set. So long as there remain unassigned large jobs, we consider the last such job to complete in $\sigma$ and create a new set in the same fashion.

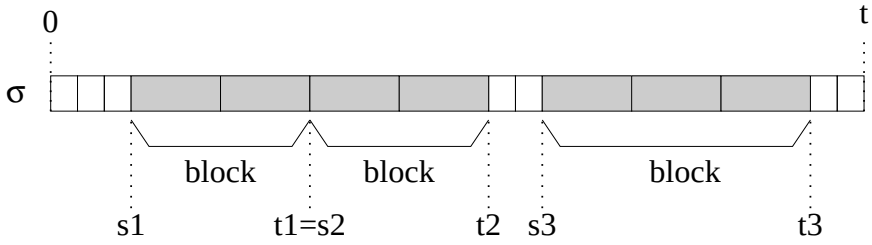


Figure 3: Analysis of GREEDY-TWOLENGTHS based on blocks.

This continues until we have assigned all large jobs which ran in $\sigma$. Figure 3 shows how the large jobs run by $\sigma$ are used as the template for defining these sets. For those large jobs which were not completed in $\sigma$, we consider the time at which they expired. We claim that such a job $J$ must have expired while a large job $J'$ was being processed in $\sigma$. Since $\kappa \geq \frac{1}{\Delta}$, each large job has a window of availability of 1 or greater. Although $J$ may arrive while a small job is being processed, that small job will complete while $J$ is still available. From that point on, $\sigma$ will not begin any other small jobs so long as $J$ is available. Therefore some other large job $J'$ must be running when $J$ expires, and we will assign $J$ to the same set to which job $J'$ was assigned.

It is clear that we have assigned all large jobs of $\mathcal{I}$ to some such set $B_i$. What remains is to prove that the sets $B_i$ are indeed blocks. From the construction, it is clear that $\sigma$ runs only jobs of $B_i$ during $[s_i, t_i)$. Also, it is clear that all jobs have expiration time during the interval $[s_i, t_i)$, with the possible exception of the job started at $s_i$. Our construction assures that jobs of $B_i \cap \sigma$ expire in this interval, and jobs of $B_i - \sigma$ are assigned to the set exactly because their expiration times lie in this interval.

We consider two possible cases depending on whether or not there exists a job of $B_i$, other than the one started at time $s_i$, which arrives strictly before time $s_i$. If no such job exists, then $B_i$ has satisfied all properties of a regular block. Alternatively, if there exists such a job of $B_i$ which arrives strictly before $s_i$ but is not run at time $s_i$, then $B_i$ is a delayed block. We have already seen that all jobs, except possibly the first, have an expiration time in the interval $[s_i, t_i)$. Since one of these jobs arrived before $s_i$ and was available, then it must also be the case that the job which was chosen to run at time $s_i$ also has a deadline strictly before $t_i$. Secondly, $\sigma$ cannot be idle immediately before time $s_i$ as a job will have arrived prior to $s_i$ and be available. In fact, $\sigma$ must be running a small job based on our construction. The job which started at time $s_i$ has a deadline strictly less than $t_i$, and so our method for adding jobs to set $B_i$ must have stopped because it reached a large job that was not preceded by another large job. Finally, the existence of this small job in the schedule from $[s_i - 1, s_i)$ assures us that none of the large jobs of $B_i$ could have arrived on or before time $s_i - 1$, as GREEDY-TWOLENGTHS would have chosen an available large job at time $s_i - 1$. Therefore, in this second case, we see that $B_i$ is indeed a delayed block. ∎

**Theorem 17** *For $\kappa \geq \frac{1}{\Delta}$,*

$$D_2(\kappa, \Delta) \leq \left(2 + \frac{\lceil \Delta \rceil - 1}{\Delta}\right),$$

*with the upper bound achieved by* GREEDY-TWOLENGTHS.

*Proof:* We partition the jobs achieved by GREEDY-TWOLENGTHS into one of the following three distinct groups:

(G1) Large jobs in $\sigma$ which lie in a regular block $B_i$ for some $i$.

(G2) Large jobs in $\sigma$ which lie in a delayed block $B_i$ for some $i$, along with all small jobs run immediately before a delayed block.

(G3) All remaining (small) jobs in $\sigma$.

Let $\sigma^*$ be the optimal schedule for $\mathcal{I}$. We partition the jobs achieved by $\sigma^*$ into one of the following three distinct groups:

(H1) The union for all regular blocks $B_i$ of jobs of $B_i$ achieved by $\sigma^*$ or small jobs started during the block's interval $[s_i, t_i)$ by $\sigma^*$.

(H2) The union for all delayed blocks $B_i$ of jobs of $B_i$ achieved by $\sigma^*$, of small jobs started during the block's interval $[s_i, t_i)$ by $\sigma^*$, and of the small job run in $\sigma$ immediately before the block.

(H3) All remaining (small) jobs in $\sigma^*$.

We will prove our claim by showing that $\|\text{H1}\| \leq c\|\text{G1}\|$, $\|\text{H2}\| \leq c\|\text{G2}\|$, and $\|\text{H3}\| \leq c\|\text{G3}\|$, for $c = (2 + \frac{\lceil\Delta\rceil - 1}{\Delta})$.

To prove that $\|\text{H1}\| \leq c\|\text{G1}\|$, we analyze each regular block individually. For regular block $B_i$, we let $g_i = (t_i - s_i)$ denote the gain of $\sigma$ on jobs from group G1 associated with this block, and we let $h_i$ denote the gain achieved by $\sigma^*$ on jobs from group H1 associated with this block. We claim that $h_i \leq (2 + \frac{\lceil\Delta\rceil - 1}{\Delta})g_i$ for all $i$, and thus $\|\text{H1}\| \leq c\|\text{G1}\|$. By definition, a regular block has at most one job which can be started outside the interval $[s_i, t_i)$. Clearly, at most $g_i$ worth of gain can be achieved by any schedule on jobs which both start and end in this interval. The optimal schedule can achieve at most the one job which can be started outside the interval, $g_i$ gain during the interval, and gain for one additional job which starts during the interval yet ends outside the interval. This bounds the gain of the optimal schedule as $h_i \leq g_i + \Delta + \Delta$. In the case that $\sigma$ achieves two or more of the jobs of $B_i$, then $g_i \geq 2\Delta$, and so $h_i \leq 2g_i$. In the case that $\sigma$ achieves exactly one job of $B_i$, then $g_i = \Delta$. In this case, however, we see that no large jobs can start during the interval yet end strictly before $t_i$. In fact, we see that there can be at most $\lceil\Delta\rceil - 1$ small jobs which start during the interval yet end strictly before $t_i$. In this case, our bound for the optimal schedule satisfies $h_i \leq \lceil\Delta\rceil - 1 + \Delta + \Delta = (2 + \frac{\lceil\Delta\rceil - 1}{\Delta})g_i$.

To prove that $\|\text{H2}\| \leq c\|\text{G2}\|$, we analyze each delayed block individual. For delayed block $B_i$, we let $g_i = (1 + t_i - s_i)$ denote the gain of $\sigma$ on jobs from group G2 associated with this block, and we let $h_i$ denote the gain achieved by $\sigma^*$ on jobs from group H2 associated with this block. We claim that $h_i \leq 2g_i$ for all $i$, and thus $\|\text{H2}\| \leq 2\|\text{G2}\|$. By definition, a delayed block does not contain any jobs which can be started outside the interval $[s_i - 1, t_i)$, therefore we only need to account for small and large jobs started within this interval. Clearly, less than $g_i$ worth of gain can be on jobs which start and end in this interval and at most one job can start in this interval and finish after $t_i$. Including the possibility that $\sigma^*$ ran the small job which led the delayed block, $h_i \leq 1 + g_i + \Delta \leq 2g_i$ as $g_i \geq 1 + \Delta$.

Finally, we show that $\|\text{H3}\| \leq 2\|\text{G3}\|$. Specifically, $\sigma$ used the resource continuously for all periods before time $t$ that were not in a block interval. Thus $\sigma^*$ can achieve at most the same number of small jobs which start strictly before $t$ yet outside the block intervals. There may also be (late) small jobs which $\sigma^*$ starts running at time $t$ or later. However, as we have seen in earlier sections, this means that such small jobs had expiration time of $t$ or greater and thus must have been completed at some point in $\sigma$. Therefore, the number of such late jobs is bounded by the number of other small jobs which $\sigma$ completed outside of the block intervals. ∎

**Theorem 18** *For $\kappa \geq 1$,*

$$D_2(\kappa, \Delta) \leq \left(1 + \frac{\lceil \Delta \rceil}{\Delta}\right),$$

*with the upper bound achieved by* Greedy-TwoLengths.

*Proof:* Using a proof structure identical to that of Theorem 17, we note that $\|\text{H2}\| \leq 2\|\text{G2}\|$ and $\|\text{H3}\| \leq 2\|\text{G3}\|$. In analyzing each regular block $B_i$, we also saw that $h_i \leq 2g_i$ in the case where $\sigma$ achieves two or more jobs of $B_i$.

To improve the overall bound, we make use of the fact that $\kappa \geq 1$ in analyzing the case of a regular block for which $\sigma$ achieves exactly one large job. In this case we claim that set $B_i$ contains only the one job run by $\sigma$. Any other job in such a block, by definition, must arrive on or after time $s_i$ and expire strictly before time $t_i = s_i + \Delta$. The existence of such a large job directly contradicts the patience requirement $\kappa \geq 1$. Therefore, in reanalyzing this block, we find that the optimal schedule can achieve $\lceil \Delta \rceil - 1$ small jobs which start during the interval yet end strictly before $t_i$, along with one other *small* job which starts before $t_i$ and ends on or after $t_i$, together with the one large job. Therefore, $h_i \leq \lceil \Delta \rceil - 1 + 1 + \Delta = \lceil \Delta \rceil + \Delta = (1 + \frac{\lceil \Delta \rceil}{\Delta}) g_i$. ∎

# 6 Lower Bounds

## 6.1 Equal Length Jobs

We begin by presenting a lower bound for the case when all jobs lengths are equal. Our deterministic lower bound matches the upper bound given in Theorem 11 for all $\kappa$. Our randomized lower bound generalizes the lower bound of 4/3 for the case $\kappa = 0$, given by Goldman *et. al.* [10].

**Theorem 19** *For all $\kappa$,*

$$D_1(\kappa) \geq 1 + \frac{1}{\lfloor \kappa \rfloor + 1} \qquad \textit{and} \qquad R_1(\kappa) \geq 1 + \frac{1}{2\lfloor \kappa \rfloor + 3}.$$

*Proof:* Let $z = \lfloor \kappa \rfloor + 1$, and thus $\kappa = z - 2\epsilon$ for some $0 < \epsilon \leq \frac{1}{2}$. Consider the following two scenarios, consisting of $z + 1$ jobs with minimum slack $\kappa$. In the first scenario $\mathcal{S}_1$, we let job $J_1 = \langle 0, 1, z + 1 + \epsilon \rangle$ and thus with slack $z + \epsilon$. We introduce $z$ other jobs each with parameters $\langle \epsilon, 1, z + 1 - \epsilon \rangle$ and thus with slack $z - 2\epsilon = \kappa$. For this input, it is possible to schedule all $z + 1$ jobs by running the $z$ other jobs from time $[\epsilon, z + \epsilon)$ followed by job $J_1$ from $[z + \epsilon, z + 1 + \epsilon)$. However, any other schedule runs at most $z$ of the jobs.

Our second scenario, $\mathcal{S}_2$, consists of the same first job $J_1$, with $z$ other jobs each with parameters $\langle 2\epsilon, 1, z + 1 \rangle$ and thus with slack $z - 2\epsilon = \kappa$. For this input, it is possible to schedule all $z + 1$ jobs by running job $J_1$ at time $[0, 1)$, followed by the other $z$ jobs from time $[1, z + 1)$. Again, any other schedule runs at most $z$ of the jobs. Both of these scenarios are shown in Figure 4.

Notice that at time $t = 0$, both of these inputs look identical to an online algorithm. Given any deterministic algorithm $A$, we consider whether that algorithm schedules job $J_1$ at time $t = 0$. For either behavior, one of the two scenarios above will result in only $z$ tasks being scheduled, although the optimal algorithm achieves all $z + 1$ tasks. Therefore $A$'s competitive ratio is at least $\frac{z+1}{z} = 1 + \frac{1}{z} = 1 + \frac{1}{\lfloor \kappa \rfloor + 1}$.

Given any randomized algorithm $A$, there must be some fixed probability $p$ that it chooses to run job $J_1$ at time $t = 0$ for either scenario. If $p > \frac{1}{2}$, we consider scenario $\mathcal{S}_1$, otherwise we consider scenario $\mathcal{S}_2$. In either case, we are assured that at least 1/2 of the time, algorithm $A$ will fail to schedule at least one job, whereas the optimal offline algorithm will always schedule all $z + 1$ jobs.

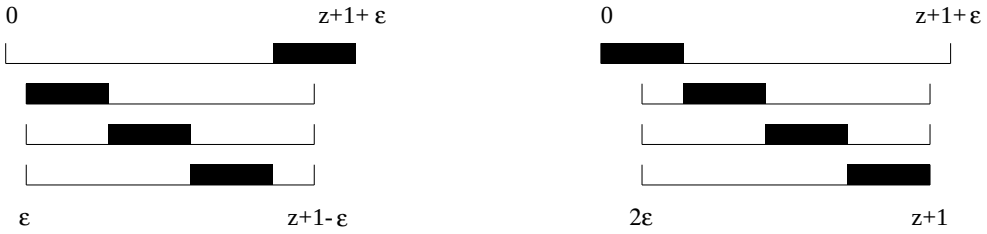


Figure 4: The left drawing shows arrivals and deadlines for instance $\mathcal{S}_1$, in which an optimal schedule must complete job $J_1$ last. The right drawing shows instance $\mathcal{S}_2$, for which an optimal schedule must complete job $J_1$ immediately.

Therefore, algorithm $A$ has an expected gain of at most $\frac{1}{2} \cdot z + \frac{1}{2} \cdot (z+1) = z + \frac{1}{2}$. The competitive ratio must be at least $\frac{z+1}{z+\frac{1}{2}} = 1 + \frac{1}{2z+1} = 1 + \frac{1}{2\lfloor \kappa \rfloor + 3}$. ∎

### 6.2 Jobs of Two Distinct Lengths

All of our remaining lower bounds apply only to deterministic algorithms. In each construction, the online algorithm is given a single initial job with an arbitrarily large deadline. For deterministic lower bounds, an adversary knows exactly how an algorithm will behave if presented with only this job. If the algorithm never schedules the job, it will have a gain of zero versus the positive gain of the optimal schedule. Alternatively, if the deterministic algorithm starts this job at some time $s$, an adversary can present additional jobs immediately after this point. All of our constructions will use fundamental instances as seen in Section 5.2.

In this section, we consider the case where all jobs are either of length 1 or length $\Delta > 1$, and where the slack for any job of length $\|J\|$ is equal to at least $\kappa\|J\|$. We refer to jobs of length 1 as "small" jobs, and jobs of length $\Delta$ as "large" jobs.

**Theorem 20** *If* $\kappa < \frac{1}{\Delta}$,

$$D_2(\kappa, \Delta) \geq 1 + \Delta.$$

*Proof:* Faced only with job $J_1 = \langle 0, 1, \infty \rangle$, we assume algorithm $A$ begins processing at time $s$. We let $\epsilon = \frac{1-\kappa\Delta}{2} > 0$ and consider an additional job $J_2 = \langle s+\epsilon, \Delta, s+\Delta+1-\epsilon \rangle$. Notice that $J_2$ has slack $1 - 2\epsilon = \kappa\Delta$, and thus the minimum slack requirement is met.

The expiration time for $J_2$ is equal to $s+1-\epsilon$, and so $A$ misses this job as it runs job $J_1$ from $[s, s+1)$. Therefore the competitive ratio for any such algorithm is $\frac{1+\Delta}{1} = 1 + \Delta$. ∎

**Theorem 21** *If* $\kappa < 1$,

$$D_2(\kappa, \Delta) \geq 2 + \frac{\lceil \Delta \rceil - 1}{\Delta}.$$

*Proof:* Faced only with job $J_1 = \langle 0, \Delta, \infty \rangle$, we assume algorithm $A$ begins processing at time $s$. We let $\epsilon = \frac{1}{2}\min(\Delta+1-\lceil\Delta\rceil, (1-\kappa)\Delta) > 0$ and consider an additional large job $J_2 = \langle s+\epsilon, \Delta, s+2\Delta-\epsilon \rangle$, as well as $\lceil\Delta\rceil - 1$ additional small jobs, each with parameters $\langle s+\epsilon, 1, s+1+\Delta-\epsilon \rangle$. Notice that the slack of all new jobs is $\Delta - 2\epsilon \geq \kappa\Delta$, and thus the minimum slack requirement is met for all of our jobs. This construction is pictured in Figure 5.

The expiration time for new jobs is equal to $s+\Delta-\epsilon$, and so $A$ will miss them all as it runs $J_1$ from $[s, s+\Delta)$. The optimal schedule achieves all jobs by running the small jobs from $[s+\epsilon, s+\epsilon+\lceil\Delta\rceil)$. Therefore the competitive ratio for any such algorithm $A$ is $\frac{2\Delta+\lceil\Delta\rceil-1}{\Delta} = 2 + \frac{\lceil\Delta\rceil-1}{\Delta}$. ∎

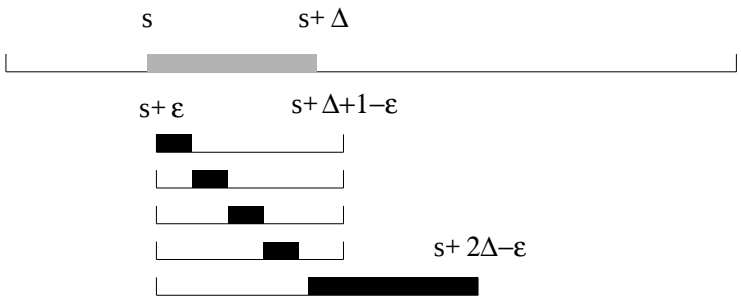


Figure 5: Lower bound construction from Theorem 21.

**Theorem 22** *If* $\kappa < \Delta$,

$$D_2(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\Delta}.$$

*Proof:* Faced only with job $J_1 = \langle 0, \Delta, \infty \rangle$, we assume algorithm $A$ begins processing at time $s$. We let $\epsilon = \frac{1}{2}\min(\Delta - \kappa, \Delta + 1 - \lceil \Delta \rceil) > 0$ and consider an additional $\lceil \Delta \rceil$ small jobs, each with parameters $\langle s + \epsilon, 1, s + 1 + \Delta - \epsilon \rangle$. Notice that the slack of the small jobs is $\Delta - 2\epsilon \geq \kappa$, and thus the minimum slack requirement is met for all of our jobs.

The expiration time for each of the new small jobs is equal to $s + \Delta - \epsilon$, and so all of these jobs are lost by $A$ which runs $J_1$ from $[s, s + \Delta)$. The optimal schedule, however can run these small jobs from $[s + \epsilon, s + \epsilon + \lceil \Delta \rceil)$ followed by $J_1$. Therefore the competitive ratio for any such algorithm is $\frac{\Delta + \lceil \Delta \rceil}{\Delta} = 1 + \frac{\lceil \Delta \rceil}{\Delta}$. ∎

**Theorem 23** *If* $\kappa \geq \Delta - 1$,

$$D_2(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1}.$$

*Proof:* Faced only with job $J_1 = \langle 0, \Delta, \infty \rangle$, we assume algorithm $A$ begins processing at time $s$. We let $\epsilon = \frac{1}{2}\min(\lfloor \kappa - \Delta \rfloor + 1 - (\kappa - \Delta),\ \Delta + 1 - \lceil \Delta \rceil) > 0$ and consider an additional $\lceil \Delta \rceil + \lfloor \kappa - \Delta \rfloor + 1$ small jobs, each with parameters $\langle s + \epsilon, 1, s + \Delta + \lfloor \kappa - \Delta \rfloor + 2 - \epsilon \rangle$. Notice that the slack of these new jobs is equal to $\Delta + \lfloor \kappa - \Delta \rfloor + 1 - 2\epsilon \geq \Delta + (\kappa - \Delta) = \kappa$, and thus the minimum slack requirement is met for all of our jobs. This construction is pictured in Figure 6.

The deadline of the small jobs is $s + \Delta + \lfloor \kappa - \Delta \rfloor + 2 - \epsilon \geq s + \lceil \Delta \rceil + \lfloor \kappa - \Delta \rfloor + 1 + \epsilon$, thus the optimal schedule can achieve all small jobs during the interval $[s + \epsilon, s + \lceil \Delta \rceil + \lfloor \kappa - \Delta \rfloor + 1 + \epsilon)$. Since $A$ runs job $J_1$ from $[s, s + \Delta)$, it can only achieve as many small jobs as can be completed within the interval $[s + \Delta, s + \Delta + \lfloor \kappa - \Delta \rfloor + 2 - \epsilon]$. As this interval has length exactly $\lfloor \kappa - \Delta \rfloor + 2 - \epsilon$, the algorithm $A$ will be able to achieve at most $\lfloor \kappa - \Delta \rfloor + 1$ small jobs. Thus, the competitiveness of any such algorithm $A$ is at least,

$$\frac{\Delta + \lceil \Delta \rceil + \lfloor \kappa - \Delta \rfloor + 1}{\Delta + \lfloor \kappa - \Delta \rfloor + 1} = 1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1}.$$

∎

### 6.3 Arbitrary Length Jobs

In this section, we consider the case where jobs can have arbitrary lengths in the range $[1, \Delta]$, and where the minimum allowable slack for any job of length $\|J\|$ is equal to $\kappa\|J\|$. It is worth noting that all of our lower bounds for $D(\kappa, \Delta)$ involve at most three distinct job lengths.

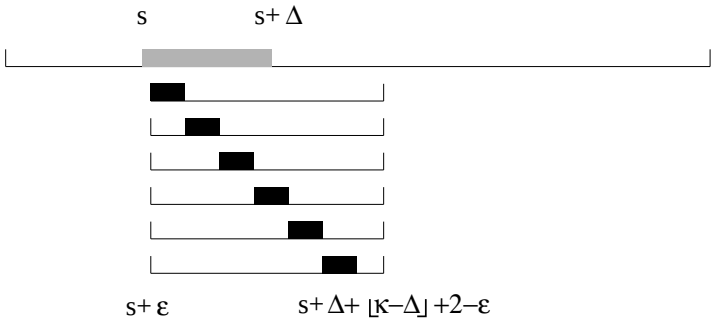


Figure 6: Lower bound construction from Theorem 23.

**Theorem 24** *If* $\kappa < \frac{1}{\Delta}$,

$$D(\kappa, \Delta) \geq 2 + \Delta\,.$$

*Proof:* Let $\epsilon$ be an arbitrarily small constant $\epsilon > 0$. Faced only with job $J_1 = \langle 0, 1, \infty \rangle$, we assume algorithm $A$ begins processing at time $s$. We consider two additional jobs $J_2 = \langle s + \epsilon, 1, s + 2 + \epsilon \rangle$ and $J_3 = \langle s + \epsilon, \Delta, s + \Delta + 1 + \epsilon \rangle$. Since $\kappa < \frac{1}{\Delta}$, the slack of $1 > \kappa\Delta$ satisfies the minimum slack requirement for both jobs.

The expiration time for both new jobs is equal to $s + 1 + \epsilon$, and so $A$ misses both $J_2$ and $J_3$ as it runs job $J_1$ from $[s, s + 1 + 2\epsilon)$. Therefore $A$ the competitive ratio for any such algorithm is $\frac{\Delta+2+\epsilon}{1+2\epsilon}$. Since $\epsilon$ can be chosen to be arbitrarily small, this lower bound can be made arbitrarily close to $2 + \Delta$. ∎

**Theorem 25** *If* $\frac{1}{\Delta} \leq \kappa \leq \lceil \Delta \rceil - 1$,

$$D(\kappa, \Delta) \geq 2 + \frac{1}{\kappa}\,.$$

*Proof:* Notice that the conditions on $\kappa$ imply that $\lceil \kappa \rceil \leq \lceil \Delta \rceil - 1 < \Delta$. We let $\epsilon$ be an arbitrarily small constant such that $0 < \epsilon \leq \frac{\Delta - \lceil \kappa \rceil}{2}$. Faced only with job $J_1 = \langle 0, \lceil \kappa \rceil + 2\epsilon, \infty \rangle$, we assume algorithm $A$ begins processing at time $s$. We consider an additional job $J_2 = \langle s + \epsilon, \frac{\lceil \kappa \rceil}{\kappa}, s + \lceil \kappa \rceil (1 + \frac{1}{k}) + \epsilon \rangle$, as well as $\lceil \kappa \rceil$ additional unit-length jobs, each with parameters $\langle s + \epsilon, 1, s + 1 + \lceil \kappa \rceil + \epsilon \rangle$. Notice that all jobs have length in the range $[1, \Delta]$ and satisfy the minimum slack requirement. This construction is pictured in Figure 7.

The expiration time for all additional jobs is equal to $s + \lceil \kappa \rceil + \epsilon$, and so $A$ misses them all as it runs $J_1$ from $[s, s + \lceil \kappa \rceil + 2\epsilon)$. Therefore the competitive ratio for any such algorithm is $\frac{\lceil \kappa \rceil (2 + \frac{1}{\kappa}) + 2\epsilon}{\lceil \kappa \rceil + 2\epsilon}$. Since $\epsilon$ can be chosen to be arbitrarily small, this lower bound can be made arbitrarily close to $2 + \frac{1}{\kappa}$. ∎

**Theorem 26** *For all values of* $\kappa > \lceil \Delta \rceil - 1$,

$$\textit{when } 0 < \{\Delta\} \leq \{\kappa\}, \qquad D(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\lfloor \kappa \rfloor + 1}.$$

$$\textit{when } \{\Delta\} = 0 \textit{ or } \{\Delta\} > \{\kappa\}, \qquad D(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\kappa},$$

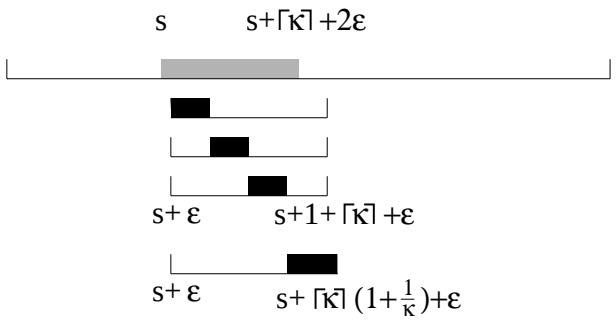


Figure 7: Lower bound construction from Theorem 25.

*Proof:* We know that $D(\kappa, \Delta) \geq D(\kappa, \Delta') \geq D_2(\kappa, \Delta')$ so long as $\Delta' \leq \Delta$.

First we consider the case that $0 < \{\Delta\} \leq \{\kappa\}$. We let $\epsilon$ be an arbitrarily small constant such that $0 < \epsilon < \{\kappa\}$ and let $\Delta' = \lceil \Delta \rceil - 1 + \epsilon$. As $\kappa \geq \Delta' - 1$, Theorem 23 assures us that,

$$\begin{aligned} D_2(\kappa, \Delta') &\geq 1 + \frac{\lceil \Delta' \rceil}{\Delta' + \lfloor \kappa - \Delta' \rfloor + 1} = 1 + \frac{\lceil \lceil \Delta \rceil - 1 + \epsilon \rceil}{\lceil \Delta \rceil - 1 + \epsilon + \lfloor \kappa - (\lceil \Delta \rceil - 1 + \epsilon) \rfloor + 1} \\ &= 1 + \frac{\lceil \Delta \rceil - 1 + \lceil \epsilon \rceil}{\lceil \Delta \rceil + \epsilon - \lceil \Delta \rceil + 1 + \lfloor \kappa - \epsilon \rfloor} = 1 + \frac{\lceil \Delta \rceil - 1 + 1}{\epsilon + \lfloor \kappa \rfloor + \lfloor \{\kappa\} - \epsilon \rfloor + 1} \\ &= 1 + \frac{\lceil \Delta \rceil}{\epsilon + \lfloor \kappa \rfloor + 1} \end{aligned}$$

As $\Delta' \leq \Delta$, we know that $D(\kappa, \Delta) \geq D_2(\kappa, \Delta')$, and since $\epsilon$ can be chosen to be arbitrarily small, we get that $D(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\lfloor \kappa \rfloor + 1}$.

In the case that $\{\Delta\} > \{\kappa\}$, let $\epsilon$ be an arbitrarily small constant such that $0 < \epsilon < \{\Delta\} - \{\kappa\}$ and in the case where $\{\Delta\} = 0$, let $\epsilon$ be an arbitrarily small constant such that $0 < \epsilon < 1 - \{\kappa\}$. Now let $\Delta' = \lceil \Delta \rceil - 1 + \{\kappa\} + \epsilon$. As $\kappa \geq \Delta' - 1$, Theorem 23 assures us that,

$$\begin{aligned} D_2(\kappa, \Delta') &\geq 1 + \frac{\lceil \Delta' \rceil}{\Delta' + \lfloor \kappa - \Delta' \rfloor + 1} = 1 + \frac{\lceil \lceil \Delta \rceil - 1 + \{\kappa\} + \epsilon \rceil}{\lceil \Delta \rceil - 1 + \{\kappa\} + \epsilon + \lfloor \kappa - (\lceil \Delta \rceil - 1 + \{\kappa\} + \epsilon) \rfloor + 1} \\ &= 1 + \frac{\lceil \Delta \rceil - 1 + \lceil \{\kappa\} + \epsilon \rceil}{\lceil \Delta \rceil - 1 + \{\kappa\} + \epsilon - \lceil \Delta \rceil + 1 + \lfloor \kappa - \{\kappa\} - \epsilon \rfloor + 1} \\ &= 1 + \frac{\lceil \Delta \rceil}{\{\kappa\} + \epsilon + \lfloor \lfloor \kappa \rfloor - \epsilon \rfloor + 1} = 1 + \frac{\lceil \Delta \rceil}{\lfloor \kappa \rfloor + \{\kappa\} + \epsilon + \lfloor -\epsilon \rfloor + 1} = 1 + \frac{\lceil \Delta \rceil}{\kappa + \epsilon} \end{aligned}$$

As $\Delta' \leq \Delta$, we know that $D(\kappa, \Delta) \geq D_2(\kappa, \Delta')$, and since $\epsilon$ can be chosen to be arbitrarily small, we get that $D(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\kappa}$. ∎

**Corollary 27**
*When* $\lceil \Delta \rceil - 1 < \kappa < \Delta$,

$$D(\kappa, \Delta) \geq 2 + \frac{1}{\kappa} - \frac{\{\kappa\}}{\kappa}.$$

*When* $\Delta \leq \kappa < \Delta + 1$ *and* $0 < \{\Delta\} \leq \{\kappa\}$,

$$D(\kappa, \Delta) \geq 2.$$

*When* $\Delta \leq \kappa < \Delta + 1$ *and either* $\{\Delta\} = 0$ *or* $\{\Delta\} > \{\kappa\}$,

$$D(\kappa, \Delta) \geq 2 - \frac{\{\kappa\}}{\kappa}.$$

*Proof:* When $\lceil \Delta \rceil - 1 < \kappa < \Delta$, we find that $\lceil \Delta \rceil - 1 = \lfloor \kappa \rfloor$ and that either $\{\Delta\} = 0$ or $\{\Delta\} > \{\kappa\}$. Theorem 26 assures us in this case that,

$$D(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\kappa} = 1 + \frac{1 + \lfloor \kappa \rfloor}{\kappa} = 1 + \frac{1 + \kappa - \{\kappa\}}{\kappa} = 2 + \frac{1}{\kappa} - \frac{\{\kappa\}}{\kappa}.$$

When $\Delta \leq \kappa < \Delta + 1$ and $0 < \{\Delta\} \leq \{\kappa\}$, we find that $\lceil \Delta \rceil = \lceil \kappa \rceil = \lfloor \kappa \rfloor + 1$. In this case, Theorem 26 assures us that,

$$D(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\lfloor \kappa \rfloor + 1} = 1 + \frac{\lfloor \kappa \rfloor + 1}{\lfloor \kappa \rfloor + 1} = 2.$$

When $\Delta \leq \kappa < \Delta + 1$ and either $\{\Delta\} = 0$ or $\{\Delta\} > \{\kappa\}$, we find that $\lceil \Delta \rceil = \lfloor \kappa \rfloor = \kappa - \{\kappa\}$. In this case, Theorem 26 assures us that,

$$D(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\kappa} = 1 + \frac{\kappa - \{\kappa\}}{\kappa} = 2 - \frac{\{\kappa\}}{\kappa}.$$

∎

# 7 One Parameter Bounds

Table 4 summarized several cross-sections of our earlier results by viewing the upper and lower limits on the competitiveness when one of the parameters is fixed.

Those results are based on the general observations that $D(\kappa', \Delta) \leq D(\kappa, \Delta)$ for $\kappa' > \kappa$ and $D(\kappa, \Delta') \leq D(\kappa, \Delta)$ for $\Delta' < \Delta$. In essence, a valid instance of the problem with parameters $\kappa$ and $\Delta'$ can be used to construct a valid instance of the problem with parameters $\kappa$ and $\Delta$ for $\Delta > \Delta'$ without changing the competitive ratio. This is done by amplifying the original instance, and then introducing one additional job with a processing time that furthers the gap between the largest and smallest jobs. Similarly, an instance with parameters $\kappa'$ and $\Delta$ can be transformed into an equivalent instance with parameters $\kappa$ and $\Delta$ for $\kappa < \kappa'$. Based on this reasoning, we can conclude,

- $\inf\limits_{\kappa} D(\kappa, \Delta) = \lim\limits_{\kappa \to \infty} D(\kappa, \Delta) \leq \lim\limits_{\kappa \to \infty} 1 + \dfrac{\Delta + 1}{\kappa} = 1$, based on Theorem 14.
- $\sup\limits_{\kappa} D(\kappa, \Delta) = D(0, \Delta) = 2 + \Delta$, based on Theorems 4 and 24.
- $\inf\limits_{\Delta} D(\kappa, \Delta) = \lim\limits_{\epsilon \to 0} D(\kappa, 1 + \epsilon) \geq 1 + \dfrac{2}{\lfloor \kappa \rfloor + 1}$, based on Theorems 24, 25 and 26.
- $\sup\limits_{\Delta} D(\kappa, \Delta) = \lim\limits_{\Delta \to \infty} D(\kappa, \Delta) = 2 + \dfrac{1}{\kappa}$, based on Theorems 5 and 25.

For the special case involving two distinct job lengths, we know trivially that $D_2(\kappa, \Delta) \leq D(\kappa, \Delta)$. It is also clear that $D_2(\kappa', \Delta) \leq D_2(\kappa, \Delta)$ for $\kappa' > \kappa$. Based on this reasoning, we can conclude,

- $\inf\limits_{\kappa} D_2(\kappa, \Delta) \leq \inf\limits_{\kappa} D(\kappa, \Delta) = 1$, from above.

- $\sup_{\kappa} D_2(\kappa, \Delta) = D_2(0, \Delta) = \max(1 + \Delta, 2 + \frac{1}{\Delta})$, by Corollary 13 and Theorems 20 and 21.

As a technicality, we cannot make the general claim $D_2(\kappa, \Delta') \leq D_2(\kappa, \Delta)$ for $\Delta' < \Delta$ in general, as $\Delta$ is defined to be the exact ratio between the two existing lengths. That is, a construction that uses lengths 1 and $\Delta'$ for some $\Delta' < \Delta$ cannot be extended to a valid instance with lengths 1 and $\Delta$. Therefore, we conclude with the following two explicit corollaries.

**Corollary 28**

$$\inf_{\Delta} D_2(\kappa, \Delta) \geq \min\left(2, 1 + \frac{2}{\kappa + 1}\right).$$

*Proof:* If $\kappa < 1$, then we can apply Theorem 21, which assures us that $D_2(\kappa, \Delta) \geq 2 + \frac{\lceil \Delta \rceil - 1}{\Delta} > 2 = \min(2, 1 + \frac{2}{\kappa+1})$. For $\kappa \geq 1$, we consider two possible cases. For values of $\Delta$ such that $\Delta > \kappa$, Theorem 22 states that $D_2(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\Delta} \geq 2 \geq \min(2, 1 + \frac{2}{\kappa+1})$. For values of $\Delta$ such that $\Delta \leq \kappa$, Theorem 23 states that

$$D_2(\kappa, \Delta) \geq 1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1} \geq 1 + \frac{2}{\kappa + 1} = \min(2, 1 + \frac{2}{\kappa + 1}),$$

using inequalities from the proof of Corollary 13. ∎

**Corollary 29** *For $\kappa > 0$,*

$$\sup_{\Delta} D_2(\kappa, \Delta) = 1 + \max\left(\frac{\lceil \kappa \rceil + 1}{\lceil \kappa \rceil}, \frac{\lfloor \kappa \rfloor + 1}{\kappa}\right).$$

*Proof:* First we consider the situation when $0 < \kappa < 1$. In evaluating the supremum, we consider two distinct cases. If $\Delta$ is chosen such that $0 < \kappa < \frac{1}{\Delta} < 1$, then the combination of Corollary 13 with Theorems 20 and 21 states that $D_2(\kappa, \Delta) = \max(1 + \Delta, 2 + \frac{1}{\Delta})$. The expression $1 + \Delta$ can be maximized as $\Delta$ approach $\frac{1}{\kappa}$ from below, in which case the limit is $1 + \frac{1}{\kappa}$. The expression $2 + \frac{1}{\Delta}$ can be maximized as $\Delta$ approaches 1 from above, in which case the limit is 3. Alternatively, if $\frac{1}{\Delta} \leq \kappa < 1$, then the combination of Theorems 17 and 21 state that $D_2(\kappa, \Delta) = \left(2 + \frac{\lceil \Delta \rceil - 1}{\Delta}\right)$. The supremum of this expression is equal to 3. Overall, for the case when $0 < \kappa < 1$, we find,

$$\sup_{\Delta} D_2(\kappa, \Delta) = \max\left(3, 1 + \frac{1}{\kappa}\right) = 1 + \max\left(\frac{1+1}{1}, \frac{0+1}{\kappa}\right) = 1 + \max\left(\frac{\lceil \kappa \rceil + 1}{\lceil \kappa \rceil}, \frac{\lfloor \kappa \rfloor + 1}{\kappa}\right).$$

Secondly, when $\kappa \geq 1$, we again consider two distinct cases based on the possible choices for $\Delta$. If $\Delta > \kappa$ is chosen, then the combination of Theorems 18 and 22 states that $D_2(\kappa, \Delta) = 1 + \frac{\lceil \Delta \rceil}{\Delta}$. We will soon claim that the supremum of this expression, contingent on condition $\Delta > \kappa$, will be achieved either for $\Delta = \lceil \kappa \rceil + \epsilon$ or $\Delta = \kappa + \epsilon$ as $\epsilon$ approaches zero from above. With this claim, we find that the supremum, subject to $1 \leq \kappa < \Delta$, equals

$$\lim_{\epsilon \to 0} 1 + \max\left(\frac{\lceil \lceil \kappa \rceil + \epsilon \rceil}{\lceil \kappa \rceil + \epsilon}, \frac{\lceil \kappa + \epsilon \rceil}{\kappa + \epsilon}\right) = 1 + \max\left(\frac{\lceil \kappa \rceil + 1}{\lceil \kappa \rceil}, \frac{\lfloor \kappa \rfloor + 1}{\kappa}\right).$$

To justify this claim, we can examine the derivative of the expression with respect to $\Delta$. The expression's value is always decreasing, except for discontinuities when $\lceil \Delta \rceil$ increases. Contingent on $\Delta > \kappa$, it is possible that the expression is maximized for $\Delta = \kappa + \epsilon$. Alternatively, the next rise in the value is at the discontinuity caused when $\Delta = \lceil \kappa \rceil + \epsilon$. The only other candidates for the

supremum beyond this point would be the other discontinuities achieved by values of $\Delta = i + \lceil \kappa \rceil + \epsilon$ for integer $i \geq 1$. However, in each case,

$$1 + \frac{\lceil \Delta \rceil}{\Delta} = 1 + \frac{i + \lceil \lceil \kappa \rceil + \epsilon \rceil}{i + \lceil \kappa \rceil + \epsilon} = 1 + \frac{i + \lceil \kappa \rceil + 1}{i + \lceil \kappa \rceil + \epsilon} < 1 + \frac{\lceil \kappa \rceil + 1}{\lceil \kappa \rceil + \epsilon}.$$

To complete the analysis for $\kappa \geq 1$, we must also consider the possibility that $\Delta \leq \kappa$ is chosen. However in this case, Theorem 14 assures us that

$$D_2(\kappa, \Delta) \leq 2 + \frac{1}{\kappa} - \frac{1}{\Delta + 1} \leq 2 + \frac{1}{\kappa} - \frac{1}{\kappa + 1} = 2 + \frac{1}{\kappa(\kappa + 1)} \leq 2 + \frac{1}{\kappa + 1} < 2 + \frac{1}{\lceil \kappa \rceil} = 1 + \frac{\lceil \kappa \rceil + 1}{\lceil \kappa \rceil}$$

and so values of $D_2(\kappa, \Delta)$ for $\Delta \leq \kappa$ are less than the supremum achieved for $1 \leq \kappa < \Delta$. ∎

# 8 Remaining Gaps

Several gaps exist in the results summarized by Tables 2 and 3. However, each of these gaps can be made arbitrarily tight, at least for certain combinations of $\kappa$ and $\Delta$. Therefore, any improvements on reducing the gaps will likely need to involve a further splintering of the case analysis. Examining the remaining gaps,

- The gap for $D_2(\kappa, \Delta)$ when $\Delta \leq \kappa < 1 + \frac{1}{\Delta} \leq 2$ can be made arbitrarily small by setting $\Delta = 1 + \epsilon$ and $\kappa = 2 - \epsilon$. In this case,

$$\lim_{\epsilon \to 0} \left( 1 + \frac{2}{\Delta + 1} \right) = 2 = \lim_{\epsilon \to 0} \left( 2 + \frac{1}{\kappa} - \frac{1}{\Delta + 1} \right).$$

- The gap for $D_2(\kappa, \Delta)$ when $\Delta \leq \min(2, \kappa)$ and $1 + \frac{1}{\Delta} \leq \kappa$ can be made arbitrarily small by setting $\Delta = 1 + \{\kappa\} + \epsilon$. In this case,

$$\lim_{\epsilon \to 0} \left( 1 + \frac{\lceil \Delta \rceil}{\Delta + \lfloor \kappa - \Delta \rfloor + 1} \right) = 1 + \frac{2}{1 + \{\kappa\} + \lfloor \lfloor \kappa \rfloor - 1 - \epsilon \rfloor + 1} = 1 + \frac{2}{\kappa}.$$

- The gap for $D(\kappa, \Delta)$ when $\lceil \Delta \rceil - 1 < k < \Delta$ can be made arbitrarily small by setting $\kappa = \lceil \Delta \rceil - 1 + \epsilon$ for arbitrarily small $\epsilon > 0$, in which case the limit approaches $2 + \frac{1}{\kappa}$.
- The gap for $D(\kappa, \Delta)$ when $\Delta \leq \kappa < \Delta + 1$ and $0 < \{\Delta\} \leq \{\kappa\}$ can be made arbitrarily small by setting $\Delta = i + \epsilon$ and $\kappa = i + 1 - \epsilon$ for any integer $i$, and arbitrarily small $\epsilon > 0$.
- The gap for $D(\kappa, \Delta)$ when $\Delta + 1 \leq \kappa$ and $0 < \{\Delta\} \leq \{\kappa\}$ can be made arbitrarily small by setting $\Delta = i + \epsilon$ and $\kappa = i + 2 - \epsilon$ for any integer $i$, and arbitrarily small $\epsilon > 0$.
- The gap for $D(\kappa, \Delta)$ when $\Delta + 1 \leq \kappa$ and $\{\Delta\} > \{\kappa\}$ can be made arbitrarily small by setting $\Delta = i + \epsilon$ and $\kappa = i + 2$ for any integer $i$, and arbitrarily small $\epsilon > 0$.

# 9 Open Questions

As discussed in the previous section, there are some settings for which a small gap remain between our lower and upper bound. However the more significant avenues for future work are to consider the effect of patience on other generalizations of the problem we have studied, as well as on other settings for admission control.

One major goal is to better understand whether randomization can be used to improve on the deterministic results. In this paper, we have set up the notation for the study of randomized online algorithms, but for the most part our results have involved deterministic algorithms. As a first step, we point towards a remaining open question of [10] concerning the case of equal length jobs with $\kappa = 0$. They show that the GREEDY algorithm is a deterministic 2-competitive algorithm, and yet the strongest randomized lower bounds still allows for the possibility for a 4/3-competitive algorithm. We feel that closing the gap for this case is a necessary precursor to improving on the results for values of $\kappa > 0$.

The problem studied in this paper can be generalized to allow for preemption, to allow for multiple processors, or to allow each job to specify its own *benefit* parameter $w_j$ (in this paper, the payoff was implicitly equal to the length of the job). Further studies along any of these lines would be beneficial.

**Acknowledgments.** The author would like to thank Sally Goldman and Subhash Suri for many insights into their previous work on this problem, as well as Eric Torng and Sanjeev Arora for several helpful conversations, and to Sally and several anonymous referees for detailed comments on earlier drafts of this paper.

## 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. *SIAM J. Comput.*, 28(5):1806–1828, 1999.

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

[5] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Ragunathan, L. Rosier, D. Shasta, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4:125–144, 1992.

[6] S. K. Baruah and J. R. Harista. Scheduling for overload in real-time systems. *IEEE Trans. on Computers*, 46(9):1034–1039, 1997.

[7] 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.

[8] 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, Carnegie Mellon University, Pittsburgh, PA, 1995.

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

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

[11] 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.

[12] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–743, 2000.

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

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

[15] E. L. Lawler, J. K. Lenstra, A. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. Graves, A. Rinnooy Kan, and P. Zipken, editors, *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. North Holland, 1993.

[16] 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.

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

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

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