Online interval scheduling on uniformly related machines

Leah Epstein* Łukasz Jeż† Jiří Sgall‡ Rob van Stee§

August 27, 2012

Abstract

We consider online preemptive throughput scheduling of jobs with fixed starting times on m uniformly related machines, with the goal of maximizing the profit of the completed jobs. In this problem, jobs are released over time. Every job has a size and a weight associated with it. A newly released job must be either assigned to start running immediately on a machine or otherwise it is dropped. It is also possible to drop an already scheduled job, but only completed jobs contribute their weights to the profit of the algorithm.

In the most general setting, no algorithm has bounded competitive ratio, and we consider a number of standard variants. We give a full classification of the variants into cases which admit constant competitive ratio (weighted and unweighted unit jobs, and C-benevolent instances, which is a wide class of instances containing proportional-weight jobs), and cases which admit only a linear competitive ratio (unweighted jobs and D-benevolent instances). In particular, we give a lower bound of m on the competitive ratio for scheduling unit weight jobs with varying sizes, which is tight. For unit size and weight we show that a natural greedy algorithm is 4/3-competitive and optimal on $m = 2$ machines, while for a large m, its competitive ratio is between 1.56 and 2. Furthermore, no algorithm is better than 1.5-competitive.

*Department of Mathematics, University of Haifa, 31905 Haifa, Israel. lea@math.haifa.ac.il.

†Institute of Computer Science, University of Wrocław, Wrocław, Poland, and Institute of Mathematics, Academy of Sciences of the Czech Republic. Research supported by MNiSW grant N N206 368839, 2010–2013, and grant IAA100190902 of GA AV ČR. lje@cs.uni.wroc.pl.

‡Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, CZ-11800 Praha 1, Czech Republic. Partially supported by the Center of Excellence – Inst. for Theor. Comp. Sci., Prague (project P202/12/G061 of GA ČR) and grant IAA100190902 of GA AV ČR. Email: sgall@iuuk.mff.cuni.cz.

§Max Planck Institute for Informatics, Saarbrücken, Germany. vanstee@mpi-inf.mpg.de.

1 Introduction

Interval scheduling is a well-studied problem with many applications, for instance work planning for personnel, call control and bandwidth allocation in communication channels [1, 2]. In this paper, we consider online interval scheduling on uniformly related machines. In this problem, jobs with fixed starting times are released online to be scheduled on m machines. Each job needs to start immediately or else be rejected. The completion time of an job is determined by its length and the speed of a machine. Problems like these occur when jobs or material should be processed immediately upon release, but there are different machines available for processing, for instance in a large factory where machines of different generations are used side by side [12].

We consider the preemptive version of this problem, where jobs can be preempted (and hence lost) at any time (for example, if more valuable jobs are released later). Without preemption, it is easy to see that no online algorithm can be competitive for most models. The only exception is the simplest version of this problem, where all jobs have unit size and weight. For this case, preemption is not needed.

On a single machine and also on parallel identical machines, it is straightforward to solve the case of unit size and weight to optimality in an online fashion using a greedy algorithm. However, for related machines it is not possible to give a 1-competitive online algorithm, since it is in general not clear which job should be run on which machine: this may depend on the timing of future arrivals. We give a lower bound of $(3 \cdot 2^{m-1} - 2)/(2^m - 1)$ on the competitive ratio for this case, which for large m tends to $3/2$ from below. We show that a simple greedy algorithm is 2-competitive and we use a more complicated lower bound construction to show that it is not better than 1.56-competitive for large m. For $m = 2$ machines, we show that it is 4/3-competitive, matching the lower bound.

Next, we consider two extensions of this model: weighted unit-sized jobs and a model where the weight of a job is determined by a fixed function of its size. This last model includes the important case of proportional weights. A function $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ (where $\mathbb{R}_0^+$ denotes the non-negative reals) is C-benevolent if it is convex, $f(0) = 0$, and $f(p) > 0$ for all $p > 0$. This implies in particular that f is continuous in $(0, \infty)$, and monotonically non-decreasing. We consider instances, called C-benevolent, where the weights of jobs are given by a fixed C-benevolent function f of their sizes. If $f(x) = ax$ for some $a > 0$, the weights are proportional. We give a 4-competitive algorithm, which can be used both for f-benevolent jobs and for weighted unit-sized jobs. This generalizes the results of Woeginger [14] for these models on a single machine.

Finally, we give a lower bound of m for unit-weight variable-sized jobs and we show that this bound is tight. A function f is D-benevolent if it is decreasing on $(0, \infty)$, $f(0) = 0$, and $f(p) > 0$ for all $p > 0$. This is a generalization of the unit-weight variable-sized jobs case, hence also for this model we have a lower bound of m. Note that in contrast, C-benevolent functions are not a generalization of unit weights and variable sizes, because the constraint $f(0) = 0$ together with convexity implies that $f(cx) \geq c \cdot f(x)$ for all $c > 0, x > 0$, so the weight is at least a linear function of the size.

For the most general model where both the weight and the size of a job can be arbitrary, it is known that no (randomized) algorithm can be competitive, already on one machine [14, 2]. For completeness, we formally extend this result to related machines (see Appendix A). This result was also mentioned (for identical machines) by Canetti and Irani [2]. For one machine, it is possible to give an $O(1)$-competitive algorithm, and even a 1-competitive algorithm, using constant resource augmentation on the speed; that is, the machine of the online algorithm is $O(1)$ times faster than

size, weight	One machine			Two related machines		m related machines	
	LB det.	UB det.	UB rand.	LB	UB	LB	UB
$1, 1$	1	1 [3, 5]	1 [3, 5]	4/3	4/3	$\frac{3 \cdot 2^{m-1}-2}{2^m-1}$	2
1, variable	4 [14]	4 [14]	2 [7]	2 [8]	4	1.693 [4, 6]	4
variable, 1	1	1 [3, 5]	1 [3, 5]	2	2	m	m
variable, D-benevolent	3 [14][1]	4 [14]	2 [9]	2	8	m	$4m$
variable, C-benevolent	4 [14]	4 [14]	2 [9]	1.693 [4, 6]	4	1.693 [4, 6]	4
variable, proportional	4 [14]	4 [14]	2 [9]	1.693 [4, 6]	4	1.693 [4, 6]	4
variable, variable	∞ [14]	$-$	$-$	∞	$-$	∞	$-$

Table 1: An overview of old and new results.

the machine of the offline algorithm that it is compared to [10, 11].

We give an overview of our results and the known results in Table 1. In this table, a lower bound for a class of functions means that there *exists* at least one function in the class for which the lower bound holds.

Previous work Faigle and Nawijn [5] and Carlisle and Lloyd [3] considered the version of jobs with unit weights on m identical machines. They gave a 1-competitive algorithm for this problem. Woeginger [14] gave optimal 4-competitive algorithms for unit sized jobs with weights, D-benevolent jobs, and C-benevolent jobs a single machine. He also showed that no online algorithm can be competitive for the general problem (with arbitrary weights and sizes) on one machine.

For unit sized jobs with weights, Fung et al. [8] gave a 3.59-competitive *randomized* algorithm for one and two (identical) machines, as well as a deterministic lower bound of 2 for two identical machines. The upper bound for one machine was improved to 2 by the same authors [7] and later generalized to the other nontrivial models [9]. See [4, 13] for additional earlier randomized algorithms. A randomized lower bound of 1.693 for one machine was given by Epstein and Levin [4]; Fung et al. [6] point out that it also holds for parallel machines.

Fung et al. [6] considered m identical machines (not shown in the table) and gave a 2-competitive algorithm for even m and a $(2 + 2/(2m - 1))$-competitive algorithm for odd $m \geq 3$. Krumke et al. [12] were the first to study these problems for uniformly related machines.

Paper organization We consider the most basic model of unit weights and sizes in Section 2 and we give our constant competitive algorithm for unit jobs with weights and C-benevolent jobs (which includes proportional jobs) in Section 3. Finally, we give a lower bound of m for unit-weight jobs in Section 4, and we show that this is tight.

For completeness, we give an unbounded lower bound for the most general case in Appendix A, which also holds for randomized algorithms. This results motivates the study of special cases.

Notation There are m machines, $M_1, M_2 \ldots, M_m$, in order of non-increasing speed. Their speeds, all no larger than 1, are denoted $s_1, s_2 \ldots, s_m$ respectively. For an instance I and algorithm ALG,

[1] This lower bound holds for all surjective functions.

ALG(I) and OPT(I) denote the number of jobs completed by ALG and an optimal schedule, respectively. The algorithm is R-competitive if $\text{OPT}(I) \leq R \cdot \text{ALG}(I)$ for every instance I.

For a job j, we denote its size by $p(j)$, its release date by $r(j)$, and its weight by $w(j) > 0$; in Section 3 and Appendix D jobs are denoted by capital J's. Any job that an algorithm runs is executed in a half-open interval $[r, d)$, where $r = r(j)$ and d is the time at which the job completes or is preempted. We call such intervals *job intervals*. If a job (or a part of a job) of size p is run on machine M_i then $d = r + \frac{p}{s_i}$. A machine is called *idle* if it is not running any job, otherwise it is *busy*.

2 Unit sizes and weights

In this section we consider the case of equal jobs, i.e., all the weights are equal to 1 and also the size of each job is 1. We first note that it is easy to design a 2-competitive algorithm, and for 2 machines we find an upper bound of 4/3 for a natural greedy algorithm.

The main results of this section are the lower bounds. First we prove that no online algorithm on m machines can be better than $(3 \cdot 2^{m-1} - 2)/(2^m - 1)$-competitive. This matches the upper bound of 4/3 for $m = 2$ and tends to 1.5 from below for $m \to \infty$. For GREEDY on $m = 3n$ machines we show a larger lower bound of $(25 \cdot 2^{n-2} - 6)/(2^{n+2} - 3)$, which tends to $25/16 = 1.5625$ from below. Thus, somewhat surprisingly, GREEDY is not 1.5-competitive.

2.1 Greedy algorithms and upper bounds

As noted in the introduction, in this case preemptions are not necessary. We may furthermore assume that whenever a job arrives and there is an idle machine, the job is assigned to some idle machine. We call such an algorithm *greedy-like*.

It can be easily observed that every greedy-like algorithm is 2-competitive: Upon arrival of a job j that a fixed optimal schedule completes, charge j to itself if ALG also schedules (and completes) j; otherwise charge j to the job ALG is running on the machine where the optimal schedule assigns j. Every ALG's job receives at most one charge of either kind, thus it schedules at least one half of the number of jobs that the optimum schedules.

We also note that some of these algorithms are indeed no better than 2-competitive: If there is one machine with speed 1 and the remaining $m - 1$ have speeds smaller than $\frac{1}{m}$, an algorithm that assigns an incoming job to a slow machine whenever possible has competitive ratio no smaller than $2 - \frac{1}{m}$. To see this consider an instance in which $m - 1$ successive jobs are released, the i-th of them at time $i - 1$, followed by m jobs all released at time m. It is possible to complete them all by assigning the first $m - 1$ jobs to the fast machine, and then the remaining m jobs each to a unique machine. However, the algorithm in question will not complete any of the first $m - 1$ jobs before the remaining m are released, so it will complete exactly m jobs.

Algorithm GREEDY: Upon arrival of a new job: If some machine is idle, schedule the job on the fastest idle machine. Otherwise reject it.

While we cannot show that GREEDY is better than 2-competitive in general, we think it is a good candidate for such an algorithm. We support this by showing that it is optimal for $m = 2$.

Theorem 2.1. GREEDY *is* 4/3*-competitive algorithm for interval scheduling of unit size and weight jobs on 2 related machines.*

Proof. Consider a schedule of GREEDY and split it into independent intervals $[R_i, D_i)$ as follows. Let R_1 be the first release time. Given R_i, let D_i be the first time after R_i when both machines are available, i.e., each machine is either idle or just started a new job. Given D_i, let R_{i+1} be the first release time larger than or equal to D_i. Note that no job is released in the interval $[D_i, R_{i+1})$. Thus it is sufficient to show that during each $[R_i, D_i)$, the optimal schedule starts at most 4/3 times the number of jobs that GREEDY does.

At any time that a job j arrives in (R_i, D_i), both machines are busy in the schedule of GREEDY, as otherwise GREEDY could schedule it. An exception could be the case when GREEDY indeed scheduled j and one machine is idle, but then j's release time would be chosen as D_i. Thus any job that ADV starts in (R_i, D_i) can be assigned to the most recent job that GREEDY started on the same machine (possibly to itself). We get a one-to-one assignment between the jobs of ADV that arrive in (R_i, D_i), and the jobs of GREEDY that arrive in $[R_i, D_i)$. Hence the optimal schedule completes at most one additional job (the one started at time R_i on the idle machine). This proves the claim if GREEDY starts at least 3 jobs in $[R_i, D_i)$.

If GREEDY starts only one job in $[R_i, D_i)$, then so does the optimal schedule. If GREEDY starts two jobs in $[R_i, D_i)$, then the first job is started on M_1. No job is released in $[R_i, D_i)$ at or after the completion of the first job on the fast machine, as GREEDY would have scheduled it. It follows that the optimal schedule cannot schedule any two of the released jobs on the same machine and schedules also only two jobs. This completes the proof. □

2.2 Lower bounds

For the first construction, we have m machines with geometrically decreasing speeds. The instance has two sets of jobs. The first part, I_m, is the set of jobs that both the algorithm and the adversary complete. The other part, E_m, consists of jobs that are completed only by the adversary.

Intuitively, the set I_m can be described recursively. It contains one *leading* job j_m to be run on M_m plus two copies of I_{m-1}. One copy is aligned so that it finishes at the same time as j_m on M_m. The other is approximately aligned with the start of the job on M_m; we offset its release times forward so that $m-1$ of its jobs are released before j_m. Because of the offsets of the release times, GREEDY runs the leading jobs $j_1, j_2, \ldots, j_m$ (where j_i is the leading job from the leftmost I_i appearing in the recursion tree) on $M_1, M_2, \ldots, M_m$, respectively. The adversary starts by scheduling the first m released jobs on different machines, cyclically shifted, so that one of them, namely the one running on M_1 finishes later than in GREEDY but the remaining $m-1$ finish earlier. Upon their completion a job from E_m is released and scheduled by the adversary; the times are arranged so that at the time of release of any job from E_m, all the machines are busy in the schedule of GREEDY. This construction for $k = 3$ is illustrated in Figure 1. Note that later the proofs use a more convenient decomposition of the binary recursion tree: I_k consists of all the leading jobs and subinstances $I_1, I_2, \ldots, I_{k-1}$.

The same idea works for a general algorithm in place of GREEDY, but we need to be more careful. We describe an adversary strategy that dynamically determines the instance. The algorithm can use an arbitrary permutation to schedule the leading jobs. We let the adversary cyclically shift the jobs, so that on $m-1$ machines they start a little bit earlier than in the algorithm's assignment. However, this slightly disturbs the timing at the end of the subinstances, so that we cannot align them exactly. To overcome this, we need to change the offsets of the leading jobs, making them geometrically decreasing in the nested subinstances, and adjust the timing on the subinstances carefully depending on the actual schedule. The details of the construction are somewhat tedious

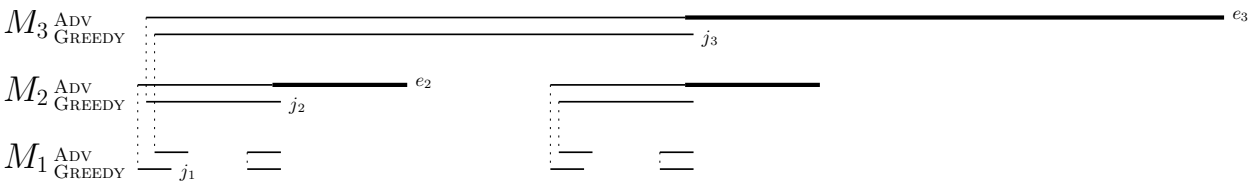

Figure 1: The instance (I_3, E_3) and (I'_1, E'_1) for GREEDY. Common jobs in GREEDY and ADV schedule are matched. Jobs that only ADV completes are thicker.

and we postpone the proof of the following theorem to Appendix B.

Theorem 2.2. *Let* ALG *be an online algorithm for interval scheduling of unit size and unit weight jobs on m related machines. Then the competitive ratio of* ALG *is at least* $(3 \cdot 2^{m-1} - 2)/(2^m - 1)$.

The second lower bound is higher, however it works only for GREEDY. We observe that cyclic shift may not be the best permutation. Instead, we create triplets of machines of the same speeds and shift the jobs cyclically among the triplets, i.e., the permutation of the leading jobs has three independent cycles of length $m/3$. Only for the three fastest machines we use the previous construction as a subinstance.

Theorem 2.3. *The competitive ratio of the* GREEDY *algorithm for interval scheduling of unit size and unit weight jobs on $m = 3n$ related machines is at least* $(25 \cdot 2^{n-2} - 6)/(2^{n+2} - 3)$.

Proof (part of which is given in Appendix C): Let ADV denote the schedule of the adversary which we construct along with the instance.

Fix $m = 3n$, $n \geq 2$, the number of machines, and the speeds

$$s_i = \begin{cases} 4^{1-i} & \text{if } i \leq 3 \ , \\ 4^{-(1+\lceil i/3 \rceil)} & \text{if } i > 3 \ . \end{cases} \tag{1}$$

The set $N_k = \{M_{3k-2}, M_{3k-1}, M_{3k}\}$ is called the *k-th cluster of machines*; note that, with the exception of N_1, all machines in a cluster have the same speed. Let $\mathcal{M}_k = \{M_1, \ldots, M_k\}$ denote the set of k fastest machines from $\mathcal{M}$ and let $\mathcal{N}_k = \{N_1, \ldots, N_k\}$ denote the set of k fastest clusters. Let $\varepsilon = 1/m$. Note that $(k-1)\varepsilon < 1$ for all $k \leq m$.

To prove the bound we are going to inductively construct a sequence of instances (I'_1, E'_1), (I'_2, E'_2), $\ldots$, (I'_n, E'_n). These are roughly the instances (I_k, E_k) from the general bound constructed for GREEDY on the cluster level, i.e., N_k corresponds to M_k from the previous construction and the leading jobs are released in triples, called *batches*. There will be many occurrences of instances (I'_k, E'_k), but we do not distinguish them explicitly by notation. For each occurrence, we also give two times $R(I'_k)$ and $D(I'_k)$ that describe the interval during which the jobs I'_k are scheduled.

Whenever the construction of an occurrence of (I'_k, E'_k) is invoked, we are given $R(I'_k)$, $D(I'_k)$ and partial schedules of both GREEDY and ADV for jobs released before $R(I'_k)$ that satisfy the following preconditions:

(A') $D(I'_k) = R(I'_k) + \frac{1}{s_3} + 2\varepsilon$ if $k = 1$, and $D(I'_k) = R(I'_k) + \frac{1}{s_{3k}} + (k-1)\varepsilon$ if $k > 1$.

(B') All clusters $\mathcal{N}_k$ are idle at time $R(I'_k)$ in the schedules of both of GREEDY and of ADV. All the remaining clusters are busy in the schedule of GREEDY at time $R(I'_k)$; furthermore, every machine in each such cluster is processing a job that will not complete before $D(I'_k)$.

In particular, for $k = n$, we start the construction by setting $R(I'_n) = 0$, $D(I'_n) = R(I'_k) + \frac{1}{s_{3m}} + (m-1)\varepsilon$ guaranteeing (A'); (B') holds trivially.

Now we describe the recursive construction of (I'_k, E'_k) together with the schedules of both Greedy and Adv. The construction proceeds inductively for $k = 1, \ldots, n$, in each inductive step we verify properties (A') and (B') for every invoked construction of a subinstance (I'_i, E'_i) and prove the following claim summarizing the desired properties of our instance (I'_k, E'_k).

Claim 2.4. Any occurrence of the instance (I'_k, E'_k) and times $R(I'_k)$ and $D(I'_k)$ for $k > 1$ has the following properties:

(i) The first $3k$ jobs of I'_k are called its *leading jobs* and are denoted by $j_1, j_2, \ldots, j_{3k}$. For each i, the jobs j_{3i-2}, j_{3i-1} and j_{3i} are released at time $R(I'_k) + (i-1)\varepsilon$; together this triplet is denoted by J_i and called a *leading batch*.

(ii) The remaining jobs of I'_k are released after time $R(E'_k)+1$. Both Greedy and Adv complete all jobs from I'_k on clusters $\mathcal{N}_k$ before $D(I'_k)$. The jobs from E'_k are released before $D(I'_k)$ and Adv completes all of them on clusters $\mathcal{N}_k$; thus they are completed before time $D(I'_k) + \frac{1}{s_{3k}}$.

(iii) The last job j of I'_k scheduled on M_1 by Greedy is completed at time $D(I'_k)$. Furthermore, while j is running, all machines $\mathcal{M}_m$ are busy in Greedy's schedule.

(iv) Every job from E'_k is released at time when all machines $\mathcal{M}_m$ are busy in Greedy's schedule.

(v) I'_k consists of exactly $16 \cdot 2^{k-2} - 3$ jobs. E'_k consists of exactly $9 \cdot 2^{k-2} - 3$ jobs.

For $k = 1$, we essentially use the instance (I_3, E_3), but we give it explicitly in the variant working for Greedy, see also Figure 1. The set I'_1 contains 7 jobs released at times $R(I'_1)$ plus $0, \varepsilon, 2\varepsilon, 3+\varepsilon, 12+\varepsilon, 12+2\varepsilon, 15+2\varepsilon$ and the set E'_1 contains 3 jobs released at times $R(I'_1)$ plus $4, 16+\varepsilon, 16+\varepsilon$. The schedule of Greedy starts the jobs at the following times after $R(I'_1)$:

On M_1 at $0, 3+\varepsilon, 12+\varepsilon, 15+2\varepsilon$. On M_2 at $\varepsilon, 12+2\varepsilon$. On M_3 at 2ε.

The schedule of Adv starts the jobs at the following times after $R(I'_1)$:

On M_1 at $2\varepsilon, 3+\varepsilon, 12+2\varepsilon, 15+2\varepsilon$. On M_2 at $0, 4, 12+\varepsilon, 16+\varepsilon$. On M_3 at $\varepsilon, 16+\varepsilon$.

It is easy to verify that all properties in Claim 2.4 hold.

For $k > 1$, I'_k is constructed as follows. First, the leading batches $J_1, J_2, \ldots, J_k$ are released as described in (i). Due to (B), the machines in $\mathcal{N}_k$ are idle both for Greedy and Adv at time $R(I'_k)$. Thus Greedy assigns the leading batch J_i to cluster $\mathcal{N}_i$ (since all three jobs in the cluster are released at the same time, their assignment within the cluster does not matter); note that Greedy cannot complete any leading job before the release of J_k, as the earliest possible completion time is $R(I'_k) + 1 > R(I'_k) + (k-1)\varepsilon$ by the choice of ε. Adv schedules J_k on N_1 and J_i on N_{i+1} for $i = 1, \ldots, k-1$. For $i = 1, \ldots, k$, denote the time when Greedy (resp. Adv) completes the leading batch scheduled on N_i by $C_{\text{Greedy}}(N_i)$ (resp. $C_{\text{Adv}}(N_i)$). Note that

$$C_{\text{Greedy}}(N_i) = R(I'_k) + \frac{1}{s_{3i}} + (i-1)\varepsilon, \tag{2}$$

$$C_{\text{Adv}}(N_1) = R(I'_k) + \frac{1}{s_3} + (k-1)\varepsilon, \text{ and} \qquad C_{\text{Adv}}(N_i) = C_{\text{Greedy}}(N_i) - \varepsilon \text{ for } i > 1. \tag{3}$$

For every cluster N_i except for N_1 we release a batch of three jobs $e_{3i-2}, e_{3i-1}, e_{3i}$ at time $C_{\text{Adv}}(N_i)$; Adv schedules these jobs on N_i. Let E' denote the union of these $k-1$ batches (i.e., a set of $3k-3$ jobs). For each $i = 1, 2, \ldots, k-1$, in increasing order, construct recursively an occurrence of the instance (I'_i, E'_i), including Adv schedule, with $D(I'_i) = C_{\text{Greedy}}(N_{i+1})$ and $R(I'_i)$ chosen so

that (A') is satisfied. Finally, let $I'_k = \{j_1, j_2, \ldots, j_{3k}\} \cup I'_1 \cup \cdots \cup I'_{k-1}$ and $E'_k = E' \cup E'_1 \cup \cdots \cup E'_{k-1}$. This completes the description of (I'_k, E'_k).

We need to verify that subinstances do not interfere with each other, namely that the chosen $R(I'_i)$, $D(I'_i)$ satisfy (B') and to prove Claim 2.4 for (I'_k, E'_k). This is fairly straightforward and postponed to Appendix C.

The construction of (I'_n, E'_n) proves the theorem. In particular, (iv) implies that Greedy does not schedule any job from E'_n and (v) implies that the competitive ratio of Greedy is at least $((16 \cdot 2^{n-2} - 3) + (9 \cdot 2^{n-2} - 3))/(16 \cdot 2^{n-2} - 3) = (25 \cdot 2^{n-2} - 6)/(16 \cdot 2^{n-2} - 3)$.

3 A constant competitive algorithm for two classes of inputs

In this section we consider two types of instances. The first type are equal-sized jobs (of size 1, without loss of generality), whose weights can be arbitrary. We also consider input instances where the weights of jobs are given by a fixed C-benevolent function f of their sizes, that is, $w(J) = f((p(J))$. We call such an instance f-benevolent.

Algorithm Alg: On arrival of a new job J do the following.

1. Use an arbitrary idle machine if such a machine exists.
2. Otherwise, if no idle machines exist, preempt the job of minimum weight among the jobs running at time $r(J)$ having a weight less than $w(J)/2$ if such jobs exist.
3. If J was not scheduled in the previous steps, then reject it.

Note that we do not use the speeds in this algorithm in the sense that there is preference of slower or faster machines in any of the steps. But clearly, the eventual schedule does depend on the speeds.

Definition 3.1. A *chain* is a maximal sequence of jobs $J_1, \ldots, J_n$, that Alg runs on one machine, such that J_j is preempted when job J_{j+1} arrives ($j = 1, \ldots, n-1$).

Observation 3.2. *For a chain $J_1, \ldots, J_n$ that* Alg *runs on machine i, J_1 starts running on an idle machine, and J_n is completed by* Alg. *Let $[r_j, d_j)$ be the time interval in which J_j is run ($j = 1, \ldots, n$). Then it holds that $r_j = r(J_j)$, $d_n - r_n = p(J_n)/s_i$, and finally $d_j - r_j < p(J_j)/s_i$, and $d_j = r_{j+1}$ for $j = 1, \ldots, n-1$.*

The following observation holds due to the preemption rule.

Observation 3.3. *For a chain $J_1, \ldots, J_n$, $2w(J_j) < w(J_{j+1})$ for $1 \le j \le n-1$.*

Consider a fixed optimal offline solution Opt, which runs all its selected jobs to completion. We say that a job J which is executed by Opt is *associated* with a chain $J_1, \ldots, J_n$ if Alg runs the chain on the machine where Opt runs J and J is released while this chain is running, i.e., $r(J) \in [r(J_1), d(J_n))$.

Claim 3.4. Every job J executed by Opt such that J is not the first job of any chain of Alg is associated with some chain.

Proof. Assume that J is not associated with any chain. The machine i which is used to execute J in Opt is therefore idle at the time $r(J)$ (before J is assigned). Thus, J is assigned in step 1 (to i or to another machine), and it is the first job of a chain. □

Thus, every job run by OPT but not by ALG is associated with a chain. We assume without loss of generality that every job in the instance either belongs to a chain or is run by OPT (or both), since other jobs have no effect on ALG and on OPT.

We assign every job that OPT runs to chains of ALG. The weight of a job J is split between J and the chain that J is associated with, where one of the two parts can be zero. In particular, if ALG does not run J then the first part must be zero, and if J is not associated with a chain then the second part must be zero. The assignment is defined as follows. Consider job J with release date r which OPT runs on machine i.

1. If J is not associated with any chain, assign a weight of $w(J)$ to J.
2. If J is associated with a chain of ALG (on machine i), let J' be the job such that $r(J) \in [r(J'), d(J'))$. Assign $\min\{w(J), 2 \cdot w(J')\}$ part of J to this chain, and assign the remainder $\max\{2 \cdot w(J') - w(J), 0\}$ part to J itself.

Note that for an f-benevolent instance, multiple jobs which are associated with a chain on a machine of speed s can be released while a given job J' of that chain is running, but only the last one can have weight above $w(J')$, since all other such jobs J satisfy $r(J)+\frac{p(J)}{s} \leq d(J')$ and $r(J) \geq r(J')$, so $p(J) \leq p(J')$ and by monotonicity $w(J) \leq w(J')$. The weight of all such jobs is assigned to the chain, while the last job associated with the chain may have some weight assigned to itself, if its weight is above $2w(J')$; this can happen only if ALG runs this job on another machine. This holds since if ALG does not run J, ALG does not preempt any of the jobs it is running, including the job J' on the machine that OPT runs J on, then $w(J) \leq 2w(J')$ (and J is fully assigned to the chain it is associated with). If ALG runs a job J on the same machine as OPT, then $J = J'$ must hold, and J is completely assigned to the chain (and not assigned to itself).

For a job J that has positive weight assignment to a chain of ALG it is associated with (such that the job J' of this chain was running at time $r(J)$), we define a pseudo-job $\pi(J)$. This job has the same release date time as J and its weight is the amount of J assigned to the chain, i.e., $\min\{w(J), 2 \cdot w(J')\}$. It is said to be assigned to the same chain of ALG that J is assigned to. If the input consists of unit jobs, then the size of $\pi(J)$ is 1. If the instance is an f-benevolent instance, then the size $p(\pi(J))$ of $\pi(J)$ is such that $f(p(\pi(J))) = 2w(J')$ (since f is continuous in $(0, \infty)$, and since there are values x_1,x_2 (the sizes of J, J') such that $f(x_1) = w(J') < 2w(J')$ and $f(x_2) = w(J) > 2w(J')$ then there must exist $x_1 < x_3 < x_2$ such that $f(x_3) = 2w(J')$), and $p(\pi(J)) \leq p(J)$.

For any chain, we can compute the total weight assigned to the specific jobs of the chain (excluding the weight assignment to the entire chain).

Claim 3.5. For a chain $J_1, \ldots, J_n$ that ALG runs on machine i, the weight assigned to J_1 is at most $w(J_1)$. The weight assigned to J_k for $2 \leq k \leq n$ is at most $w(J_k) - 2w(J_{k-1})$. The total weight assigned to the jobs of the chain is at most $w(J_n) - \sum_{k=1}^{n-1} w(J_k)$.

Proof. The property for J_1 follows from the fact that the assigned weight never exceeds the weight of the job. Consider job J_k for $k > 1$. Then $w(J_k) > 2w(J_{k-1})$ by Observation 3.3. If there is a positive assignment to J_k, then the machine i' where OPT runs J_k is not i. At the time $r(J_k)$ all machines are busy (since the scheduling rule prefers idle machines, and J_k preempts J_{k-1}). Moreover, the job J' running on machine i' at time $r(J_k)$ satisfies $w(J') \geq w(J_{k-1})$. Thus J_k is assigned $w(J_k) - 2 \cdot w(J') \leq w(J_k) - 2w(J_{k-1})$. The total weight assigned to the jobs of

the chain is at most $w(J_1) + \sum_{k=2}^{n} (w(J_k) - 2w(J_{k-1})) = w(J_1) + \sum_{k=2}^{n} w(J_k) - 2\sum_{k=1}^{n-1} w(J_k) = \sum_{k=1}^{n} w(J_k) - 2\sum_{k=1}^{n-1} w(J_k) = w(J_n) - \sum_{k=1}^{n-1} w(J_k)$. □

Definition 3.6. For a given chain $J_1, \ldots, J_n$ of ALG running on machine i, an *alt-chain* is a set of pseudo-jobs $J'_1, \ldots, J'_{n'}$ such that $r(J'_k) \geq r(J'_{k-1}) + \frac{p(J'_{k-1})}{s_i}$ for $2 \leq k \leq n'$, $r(J'_1) \geq r(J_1)$, $r(J'_{n'}) < d(J'_n)$, (that is, all jobs of the alt-chain are released during the time that the chain of ALG is running, and they can all be assigned to run on machine i in this order). Moreover, if $r(J'_k) \in [r_\ell, d_\ell)$, then $w(J'_k) \leq 2 \cdot w(J_\ell)$.

Lemma 3.7. *For unit jobs, a chain $J_1, \ldots, J_n$ of* ALG *on machine i and any alt-chain $J'_1, \ldots, J'_{n'}$ satisfy*

$$\sum_{k=1}^{n'} w(J'_k) \leq \sum_{\ell=1}^{n} w(J_\ell) + 2w(J_n).$$

Proof. For every job J_ℓ, there can be at most one job of the alt-chain which is released in $[r_\ell, d_\ell)$, since the time to process a job on machine i is $\frac{1}{s_i}$ and thus difference between release times of jobs in the alt-chain is at least $\frac{1}{s_i}$, while $d_\ell \leq r_\ell + \frac{1}{s_i}$. However, every job of the alt-chain J'_k must have a job of the chain running at $r(J'_k)$. If job J'_k of the alt-chain has $r(J'_k) \in [r_\ell, d_\ell)$ then by definition $w(J'_k) \leq 2 \cdot w(J_\ell)$, which shows $\sum_{k=1}^{n'} w(J'_k) \leq 2\sum_{\ell=1}^{n} w(J_\ell)$.

Using $w(J_k) > 2w(J_{k-1})$ for $2 \leq k \leq n$ we find $w(J_k) < \frac{w(J_n)}{2^{n-k}}$ for $1 \leq k \leq n$ and $\sum_{k=1}^{n-1} w(J_k) < w(J_n)$. Thus $\sum_{k=1}^{n'} w(J'_k) \leq \sum_{\ell=1}^{n} w(J_\ell) + 2w(J_n)$. □

Lemma 3.8. *For C-benevolent instances, a chain $J_1, \ldots, J_n$ of* ALG *on machine i and any alt-chain $J'_1, \ldots, J'_{n'}$ satisfy*

$$\sum_{k=1}^{n'} w(J'_k) \leq \sum_{\ell=1}^{n} w(J_\ell) + 2w(J_n).$$

This lemma can be deduced from a claim in [14] to analyze this algorithm for one machine. For completeness, we present a detailed proof in Appendix D.

Observation 3.9. *For a chain $J_1, \ldots, J_n$ of* ALG*, the sorted list of pseudo-jobs (by release date) assigned to it is an alt-chain, and thus the total weight of pseudo-jobs assigned to it is at most $2\sum_{\ell=1}^{n} w(J_\ell)$.*

Proof. By the assignment rule, every job which is assigned to the chain (partially or completely) is released during the execution of some job of the chain. Consider a pseudo-job J assigned to the chain, and let J' be the job of the chain executed at time $r(J)$.

The pseudo-job $\pi(J)$ has weight at most $\min\{w(J), 2 \cdot w(J')\}$. Since the set of pseudo-jobs assigned to the chain results from a set of jobs that OPT runs of machine i, by possibly decreasing the sizes of some jobs, the list of pseudo-jobs can still be executed on machine i. □

Theorem 3.10. *The competitive ratio of* ALG *is at most 4 for unit length jobs, and for C-benevolent instances.*

Proof. The weight allocation partitions the total weight of all jobs between the chains, thus is it sufficient to compare the total weight a chain was assigned (to the entire chain together with

assignment to specific jobs) to the weight of the last job of the chain (the only one which ALG completes), which is $w(J_n)$.

Consider a chain $J_1, \ldots, J_n$ of ALG. The total weight assigned to it is at most $(w(J_n) - \sum_{k=1}^{n-1} w(J_k)) + (\sum_{\ell=1}^{n} w(J_\ell) + 2w(J_n)) = 4w(J_n)$. □

4 Tight bound of m for unit weights and variable sizes

It is easy to see that the following algorithm is m-competitive. Use only the fastest machine. Accept any job when this machine is idle, and only interrupt a job for an earlier finishing job. This algorithm is optimal on one machine [5, 3]. That is, it selects a subset S of the jobs in the input which maximizes the profit on the fastest machine. In the optimal schedule for m machines, no set assigned to *any* machine can be more valuable than S because this set could also have been processed by the fastest machine, contradicting the definition of S. This proves the competitive ratio of m. We now give a matching lower bound. Note that Krumke et al. [12] claimed an upper bound of 2 for this problem, which we show is incorrect.

Fix $0 < \varepsilon < \frac{1}{2}$ such that $\frac{1}{\varepsilon}$ is integer. Our goal is to show that no online algorithm can be better than $(1-\varepsilon)m$-competitive. We define $M = (\frac{1}{\varepsilon} - 1)m$ and $N = m^3 + Mm^2 + Mm$.

Input One machine is fast and has speed 1. The other $m-1$ machines have speed $1/N$. The input sequence will consist of at most N jobs, which we identify with their numbers. Job j will have size $p(j) = 2^{N-j}$ and release time $r(j) \geq j$; we let $r(1) = 1$. The input consists of phases which in turn consist of subphases. Whenever a (sub)phase ends, no jobs are released for some time in order to allow the adversary to complete its most recent job(s). ALG will only be able to complete at most one job per full phase (before the next phase starts). The time during which no jobs are released is called a *break*.

Specifically, if ALG assigns job j to a slow machine or rejects it, the adversary assigns it to the fast machine instead, and we will have $r(j+1) = r(j) + p(j)$. We call this a *short break* (of length $p(j)$). A short break ends a subphase.

If ALG assigns job j to the fast machine, then in most cases, job j is *rejected* by the adversary and we set $r(j+1) = r(j+1)$. The only exception occurs when ALG assigns m consecutive jobs to the fast machine (since at most N jobs will arrive, and $p(j) = 2^{N-j}$, each of the first $m-1$ jobs is rejected by ALG when the next job arrives). In that case, the adversary assigns the first (i.e., largest) of these m jobs to the fast machine and the others to the slow machines (one job per machine). After the m-th job is released, no further jobs are released until the adversary completes all these m jobs. The time during which no jobs are released is called a *long break*, and it ends a phase.

The input ends after there have been M long breaks, or if $m^2 + bm$ short breaks occur in total (in all phases together) before b long breaks have occurred. Thus the input always ends with a break. We show in Appendix E that if there are $m^2 + bm$ short breaks in total before the b-th long break, ALG can complete at most $b - 1 + m$ jobs from the input (one per long break plus whatever jobs it is running when the input ends), whereas OPT earns $m^2 + bm$ during the short breaks alone. This implies a ratio of m and justifies ending the input in this case (after the $(m^2 + bm)$-th short break). If the M-th long break occurs, the input also stops. ALG has completed at most M jobs and can complete at most $m-1$ more. OPT completes at least Mm jobs in total (not counting any short breaks). The ratio is greater than $Mm/(M+m) = (1-\varepsilon)m$ for $M = (\frac{1}{\varepsilon} - 1)m$.

References

[1] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. of 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*, pages 312–320, 1994.

[2] R. Canetti and S. Irani. Bounding the power of preemption in randomized scheduling. *SIAM Journal on Computing*, 27(4):993–1015, 1998.

[3] M. C. Carlisle and E. L. Lloyd. On the k-coloring of intervals. *Discrete Appliled Mathematics*, 59(3):225–235, 1995.

[4] L. Epstein and A. Levin. Improved randomized results for the interval selection problem. *Theoretical Computer Science*, 411(34-36):3129–3135, 2010.

[5] U. Faigle and W. M. Nawijn. Note on scheduling intervals on-line. *Discrete Appliled Mathematics*, 58(1):13–17, 1995.

[6] S. P. Y. Fung, C. K. Poon, and D. K. W. Yung. On-line scheduling of equal-length intervals on parallel machines. *Information Processing Letters*, 112(10):376–379, 2012.

[7] S. P. Y. Fung, C. K. Poon, and F. Zheng. Improved randomized online scheduling of unit length intervals and jobs. In *Proc. 6th Intl. Workshop Approx. Online Algorithms (WAOA2008)*, pages 53–66, 2008.

[8] S. P. Y. Fung, C. K. Poon, and F. Zheng. Online interval scheduling: randomized and multiprocessor cases. *Journal of Combinatorial Optimization*, 16(3):248–262, 2008.

[9] S. P. Y. Fung, C. K. Poon, and F. Zheng. Improved randomized online scheduling of intervals and jobs. *CoRR*, abs/1202.2933, 2012.

[10] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[11] C.-Y. Koo, T. W. Lam, T.-W. Ngan, K. Sadakane, and K.-K. To. On-line scheduling with tight deadlines. *Theoretical Computer Science*, 295:251–261, 2003.

[12] S. O. Krumke, C. Thielen, and S. Westphal. Interval scheduling on related machines. *Computers & Operations Research*, 38(12):1836–1844, 2011.

[13] S. S. Seiden. Randomized online interval scheduling. *Operations Research Letters*, 22(4-5):171–177, 1998.

[14] G. J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130(1):5–16, 1994.

A Variable weights and sizes

We show that for the most general setting, no competitive algorithm can exist (not even a randomized one).

Proposition A.1. *The competitive ratio of every randomized algorithm for variable lengths and variable weights is unbounded.*

Proof. Let ALG_m be an arbitrary randomized preemptive algorithm for m machines. Let C be an arbitrary constant, and let $C' = mC$. Define the following algorithm ALG for one machine of speed s_1: ALG chooses an integer i in $\{1, 2, ..., m\}$ uniformly with probability $1/m$ and acts as ALG_m acts on machine i. Since the speed of i is at most s_1, this is possible. Note that ALG is randomized even if ALG_m is deterministic. For every input J, $\mathbb{E}(\text{ALG}(J)) = \frac{1}{m} \cdot \mathbb{E}(\text{ALG}_m(J))$.

Let OPT_m denote an optimal solution for m machines, and OPT_1 on one machine. Clearly $\text{OPT}_1(J) \leq \text{OPT}_m(J)$ for every input J. Let I be an input such that $\text{OPT}_1(I) \geq C' \cdot \mathbb{E}(\text{ALG}(I))$ (its existence is guaranteed, since ALG's competitive ratio is unbounded [2]). Then

$$C \cdot \mathbb{E}(\text{ALG}_m(I)) = mC \cdot \mathbb{E}(\text{ALG}(I)) \leq \text{OPT}_1(I) \leq \text{OPT}_m(I).$$

Thus the competitive ratio of ALG_m is unbounded. □

B General Lower Bound for Unit Weights and Times

In this appendix we prove Theorem 2.2

Following Section 2.1, we may assume that ALG is greedy-like. Let ADV denote the schedule of the adversary which we construct along with the instance.

Fix m, the number of machines and the speeds $s_k = 4^{-k}$. Thus a job processed on M_k takes time $\frac{1}{s_k} = 4^k$. Let $\mathcal{M}_k = \{M_1, \ldots, M_k\}$ denote the set of k fastest machines from $\mathcal{M}$. For $k = 1, \ldots, m$, let $\varepsilon_k = m^{k-m-1}$. Note that $\varepsilon_k < 1$ and $k\varepsilon_k < m\varepsilon_k = \varepsilon_{k+1}$.

To prove the bound we are going to inductively construct a sequence of instances (I_1, E_1), (I_2, E_2), ..., (I_m, E_m). Each instance I_k is run by both ALG and ADV on machines $\mathcal{M}_k$. In fact, due to the recursive construction, there will be many occurrences of instances (I_k, E_k); we do not introduce separate notation for each of them as the particular occurrence will be clear from the context. For each occurrence, we also construct two times $R(I_k)$ and $D(I_k)$ that describe the interval during which the jobs I_k are scheduled.

Whenever the construction of an occurrence of (I_k, E_k) is invoked, we are given $R(I_k)$, $D(I_k)$ and partial schedules of both GREEDY and ADV for jobs released before $R(I_k)$ that satisfy the following preconditions:

(A) $D(I_k) = R(I_k) + \frac{1}{s_k} + (k-1)\varepsilon_k$.

(B) All machines $\mathcal{M}_k$ are idle at time $R(I_k)$ in the schedules of both of ALG and of ADV. All the remaining machines are busy in the schedule of ALG at time $R(I_k)$; furthermore, each of them is processing a job that will not complete before $D(I_k)$.

In particular, for $k = m$, we start the construction by setting $R(I_m) = 0$ and $D(I_m) = \frac{1}{s_m} + (m-1)\varepsilon_m$ guaranteeing (A); (B) holds trivially.

Now we describe the recursive construction of I_k and E_k together with the schedule of ADV and the proof of the following Claim B.1 summarizing the desired properties of our instances. The construction depends on the actual schedule of ALG before the current time; this is sound, as during the construction the current time only increases. (Note that this also means that different occurrences of (I_k, E_k) may look slightly differently.)

Claim B.1. Any occurrence of the instance (I_k, E_k) and times $R(I_k)$ and $D(I_k)$ has the following properties:

(i) The first k jobs of I_k are called its *leading jobs* and are denoted by $j_1, j_2, \ldots, j_k$. For each i, the job j_i is released at time $R(I_k) + (i-1)\varepsilon_k$.

(ii) The remaining jobs of I_k are released after time $R(I_k) + 1$. Both ALG and ADV complete all jobs from I_k on machines $\mathcal{M}_k$ before $D(I_k)$. The jobs from E_k are released before $D(I_k)$ and ADV completes all of them on machines $\mathcal{M}_k$; thus they are completed before time $D(I_k) + \frac{1}{s_k}$.

(iii) The last job j of I_k scheduled on M_1 by ALG is completed during time interval $[D(I_k) - k\varepsilon_k, D(I_k)]$. Furthermore, while j is running, all machines $\mathcal{M}_m$ are busy in the schedule of ALG.

(iv) Every job from E_k is released at a time when all machines $\mathcal{M}_m$ are busy in the schedule of ALG.

(v) I_k consists of exactly $2^k - 1$ jobs. E_k consists of exactly $2^{k-1} - 1$ jobs.

For $k = 1$, we set $I_1 = \{j_1\}$ where j_1 is a single leading job released at $R(I_1)$ and E_1 is empty. This instance trivially satisfies all the properties from Claim B.1; for (iii) note that j_1 is the only and last job, it is completed before $D(I_1)$ and by the assumption (B), all the other machines are running a job during the whole interval $[R(I_1), D(I_1)]$.

For $k > 1$, I_k is constructed as follows. First, the leading jobs $j_1, j_2, \ldots, j_k$ are released as described in (i). We let time proceed to the time $R(I_k) + 1$ and examine the schedule of ALG. Since later we make sure that (ii) holds, i.e., no more jobs are released before $R(I_k) + 1$, this is sound. Since for ALG the machines in $\mathcal{M}_k$ are idle at time $R(I_k)$ and all the other machines are busy, ALG assigns all the leading jobs to machines in $\mathcal{M}_k$ in one-to-one correspondence; this uses the assumption that ALG is greedy-like and also the fact that no leading job can complete before the release of j_k, as the earliest possible completion time is $R(I_k) + 1 > R(I_k) + (k-1)\varepsilon_k$ by the choice of ε_k.

For $i = 1, \ldots, k-1$, ADV schedules j_i on the machine where ALG schedules j_{i+1}; ADV schedules j_k on the machine where ALG schedules j_1. For $i = 1, \ldots, k$, denote the time when ALG (resp. ADV) completes the leading job scheduled on M_i by $C_{\mathrm{ALG}}(M_i)$ (resp. $C_{\mathrm{ADV}}(M_i)$). Note that

$$C_{\mathrm{ALG}}(M_i), C_{\mathrm{ADV}}(M_i) \in \left[R(I_k) + \frac{1}{s_i}, R(I_k) + \frac{1}{s_i} + (k-1)\varepsilon_k \right], \tag{4}$$

which means that the completion times on the machines are almost independent of the permutation of the leading jobs in each schedule and they increase fast with i.

The construction implies that, for all machines $\mathcal{M}_k$ with one exception,

$$C_{\mathrm{ALG}}(M_i) = C_{\mathrm{ADV}}(M_i) + \varepsilon_k. \tag{5}$$

(The exception is the machine where ALG schedules j_1.) For every machine M_i satisfying (5) we release a job e_i at time $C_{\mathrm{ADV}}(M_i)$. Let E denote the set of these $k-1$ jobs e_i.

For each $i = 1, 2, \ldots, k-1$, in increasing order, construct recursively an occurrence of the instance (I_i, E_i), including ADV schedule, with $R(I_i) = C_{\mathrm{ALG}}(M_{i+1}) - (i-1)\varepsilon_i - \frac{1}{s_i}$ and $D(I_i) =$

$C_{\text{ALG}}(M_{i+1})$. For the rest of the construction and of the proof of Claim B.1, let (I_i, E_i) denote this particular occurrence. Finally, let $I_k = \{j_1, j_2, \ldots, j_k\} \cup I_1 \cup \cdots \cup I_{k-1}$ and $E_k = E \cup E_1 \cup \cdots \cup E_{k-1}$. This completes the description of (I_k, E_k).

We need to verify that subinstances do not interfere with each other, namely that the chosen $R(I_i)$, $D(I_i)$ and the partial schedules of ALG and ADV satisfy (A) and (B). The property (A) follows from the choice of $R(I_i)$ and $D(I_i)$. To prove (B), we need to show that each $R(I_i)$ is sufficiently large compared to $D(I_{i-1})$ (or compared to $R(I_k)$ for $i = 1$). For $i = 1, \ldots, k-1$, using (4) and the choice of ε_i we have $D(I_i) = C_{\text{ALG}}(M_{i+1}) \in \left[R(I_k) + \frac{1}{s_{i+1}}, R(I_k) + \frac{1}{s_{i+1}} + 1\right]$ and thus $R(I_i) \geq R(I_k) + \frac{1}{s_{i+1}} - \frac{1}{s_i} - 1 = R(I_k) + 3\frac{1}{s_i} - 1$. This implies that before $R(I_i)$, the leading job on M_i is completed in both ALG and ADV schedules; also the extra job e_i (if it exists) is completed, since it is released before time $R(I_k) + \frac{1}{s_i} + 1$ and takes time $\frac{1}{s_i}$. Furthermore, for $i = 2, \ldots, k-1$, this implies that $R(I_i) > D(I_{i-1}) + \frac{1}{s_{i-1}}$, which means that all the jobs from (I_{i-1}, E_{i-1}) are completed, using (ii) for (I_{i-1}, E_{i-1}). Thus we conclude that all the jobs started before $R(I_i)$ on $\mathcal{M}_i$ are completed and these machines are idle in both schedules. The machines $M_{i+1}, \ldots, M_k$ are still processing their leading jobs at $R(I_i)$, as $R(I_i) < R(I_k) + \frac{1}{s_{i+1}}$. Finally, the machines $M_{k+1}, \ldots, M_m$ are processing jobs guaranteed by the assumption (B) for (I_k, E_k), as $R(I_i) \in [R(I_k), D(I_k)]$. This completes the proof of the assumption (B) for the subinstances (I_i, E_i).

Finally, we verify the remaining properties from Claim B.1 for (I_k, E_k). Property (i) follows from the construction.

Property (ii): The fact that non-leading jobs are released only after $R(I_k) + 1$ follows from the analysis of $R(I_1)$ in the previous paragraph. From (4) it follows that in both schedules, the leading jobs complete by time $R(I_k) + \frac{1}{s_k} + (k-1)\varepsilon_k = D(I_k)$. Furthermore, each $D(I_i)$ is equal to some completion time of a leading job, thus it is at most $D(I_k)$ and all the jobs from I_i complete by $D(I_i)$ by (ii) for (I_i, E_i). The jobs from E are released before $D(I_k)$ by construction, for the remaining jobs from E_k, i.e., those from some E_i, it follows again by (ii) for (I_i, E_i).

Property (iii): Let j be the last job of I_k scheduled on M_1 in ALG and let C be its completion time. This is the last job of I_{k-1}, so by (iii) for I_{k-1} we have $C \in [D(I_{k-1}) - (k-1)\varepsilon_{k-1}, D(I_{k-1})] \subseteq \left[C_{\text{ALG}}(M_k) - \varepsilon_k, C_{\text{ALG}}(M_k)\right]$ using $D(I_{k-1}) = C_{\text{ALG}}(M_k)$ and the choice of $\varepsilon_k > (k-1)\varepsilon_{k-1}$. By (4), we have $C_{\text{ALG}}(M_k) \geq D(I_k) - (k-1)\varepsilon_k$, thus $C \geq C_{\text{ALG}}(M_k) - \varepsilon_k \geq D(I_k) - k\varepsilon_k$. Finally, $C \leq D(I_k)$ follows from (ii).

Property (iv): First consider job e_i, $i = 1, \ldots, k$, released at time $C_{\text{ADV}}(M_i)$. Machine M_i is busy in ALG, as otherwise we do not release e_i. Machines $M_{i+1}, \ldots, M_k$ are busy with their leading jobs by (4), and machines $M_{k+1}, \ldots, M_m$ are busy by the assumption (B) for I_k. If $i > 1$, then $C_{\text{ADV}}(M_i) = D(I_{i-1}) - \varepsilon_k$. Since $(i-1)\varepsilon_{i-1} < \varepsilon_k < 1$, it follows from (iii) for I_{i-1} that at time $C_{\text{ADV}}(M_i)$ when e_i is released, the last job of I_{i-1} on M_1 is running in ALG schedule and all machines $\mathcal{M}_m$ are busy. Now consider other jobs from E_k; each such job is in some E_i, thus the statement follows from (iv) for E_i.

Property (v): We have, using the inductive assumption, $|I_k| = k + \sum_{i=1}^{k-1} |I_i| = k + \sum_{i=1}^{k-1} (2^i - 1) = 2^k - 1$ and $|E_k| = k - 1 + \sum_{i=1}^{k-1} |E_i| = k - 1 + \sum_{i=1}^{k-1} (2^{i-1} - 1) = 2^{k-1} - 1$.

The construction of (I_m, E_m) proves the theorem. In particular, (iv) implies that ALG does not schedule any job from E_m and (v) implies that the competitive ratio of ALG is at least $\left((2^m - 1) + (2^{m-1} - 1)\right)/(2^m - 1) = (3 \cdot 2^{m-1} - 2)/(2^m - 1)$.

The construction of I_k is illustrated in Figures 2 and 3.

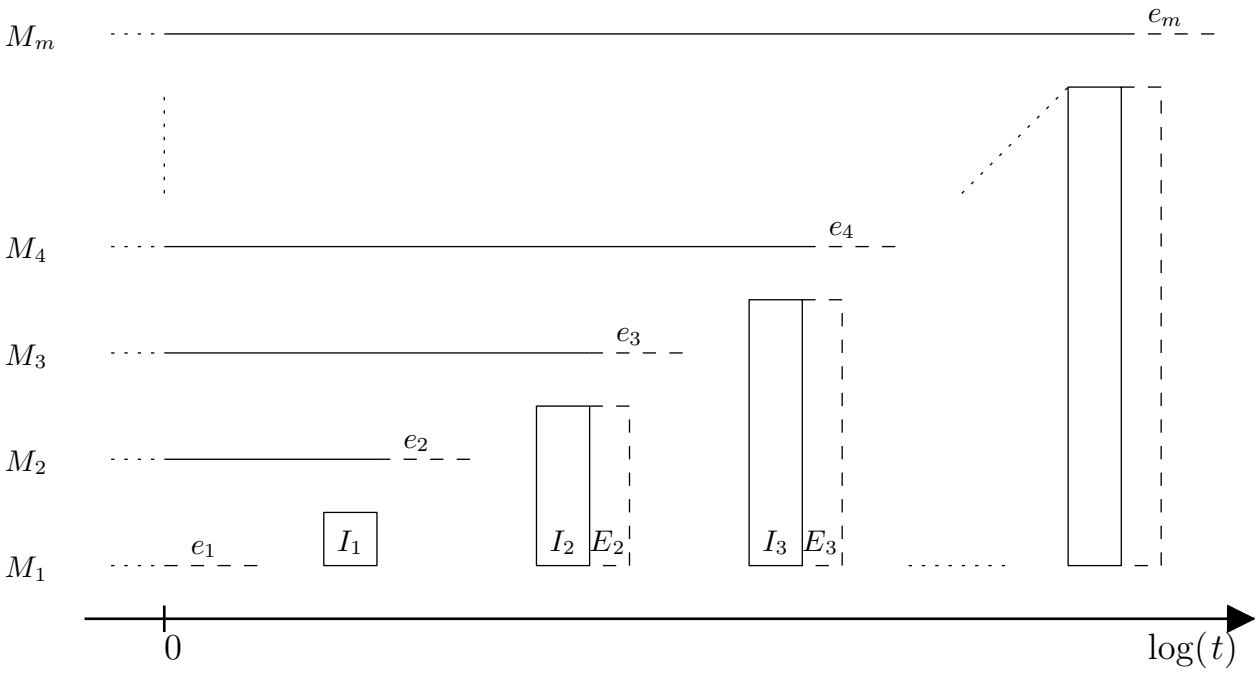

Figure 2: Overview of construction of I_k. In this figure the ϵ-s are ignored, i.e., their values are set to zero. The time scale is logarithmic and starts roughly at 1, which, with ϵ-s set to zero, equals $C_{\text{ALG}}(M_1) = C_{\text{ADV}}(M_1)$. The release times of the leading jobs are not depicted, since with ϵ-s set to zero, they equal zero. All potential extra jobs are depicted, though the one for machine where ALG schedules j_1 is not released.

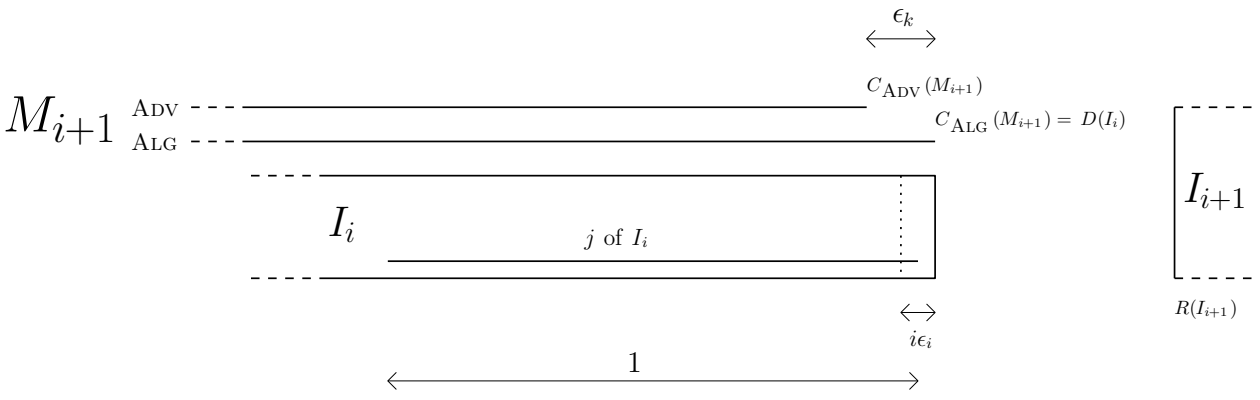

Figure 3: Details of construction of I_k for the time around the completion(s) of the leading job(s) on M_{i+1} by both ALG and ADV. In this figure the scale is roughly linear: while ϵ-s are smaller than 1, they might be enlarged in the figure.

C Lower Bound for Greedy for Unit Weights and Times

Here we complete the proof of Theorem 2.3. Namely, we verify that in the construction of (I'_k, E'_k), the subinstances do not interfere with each other, i.e., that the chosen $R(I'_i)$, $D(I'_i)$ satisfy (B') and we prove Claim 2.4 for (I'_k, E'_k).

To prove (B'), we need to show that each $R(I'_i)$ is sufficiently large compared to $D(I'_{i-1})$ (or compared to $R(I'_k)$ for $i = 1$). For $i = 1, \ldots, k-1$, using (2) we have $D(I'_i) = C_{\text{GREEDY}}(N_{i+1}) = R(I'_k) + \frac{1}{s_{3(i+1)}} + i\epsilon$ and thus $R(I'_i) = R(I'_k) + \frac{1}{s_{3(i+1)}} - \frac{1}{s_{3i}} + \epsilon = R(I'_k) + 3\frac{1}{s_{3i}} + \epsilon$. This implies that before $R(I'_i)$, the leading batch on N_i is completed in both GREEDY and ADV schedules; also the extra jobs $e_{3i-2}, e_{3e-1}, e_{3e}$ (for $i > 1$) are completed, since it is released before time $R(I'_k) + \frac{1}{s_{3i}} + (i-1)\epsilon$ and takes time $\frac{1}{s_{3i}}$. Furthermore, for $i = 2, \ldots, k-1$, this implies that $R(I'_i) > D(I'_{i-1}) + \frac{1}{s_{3(i-1)}}$, which means that all the jobs from (I'_{i-1}, E'_{i-1}) are completed, using (ii) for (I'_{i-1}, E'_{i-1}). Thus we conclude that all the jobs started before $R(I'_i)$ on $\mathcal{N}_i$ are completed and these clusters are idle in both schedules. The clusters $N_{i+1}, \ldots, N_k$ are still processing their leading batches at $R(I'_i)$, as $R(I'_i) < R(I'_k) + \frac{1}{s_{3(i+1)}}$. Finally, the clusters $N_{k+1}, \ldots, N_m$ are processing batches, by the assumption (B') for (I'_k, E'_k), since $R(I'_i) \in [R(I'_k), D(I'_k)]$. This completes the proof of the precondition (B') for the subinstances (I'_i, E'_i).

Finally, we verify Claim 2.4 for (I'_k, E'_k). Property (i) follows from the construction.

Property (ii): The fact that non-leading jobs are released only after $R(I'_k) + 1$ follows from the analysis of $R(I'_1)$ in the previous paragraph. From (2) and (3) it follows that in both schedules, the leading jobs complete by time $R(I'_k) + \frac{1}{s_{3k}} + (k-1)\varepsilon = D(I'_k)$. Furthermore, each $D(I'_i)$ is equal to some completion time of a leading batch, thus it is at most $D(I'_k)$ and all the jobs from I'_i complete by $D(I'_i)$ by (ii) for (I'_i, E'_i). The jobs from E' are released before $D(I'_k)$ by construction, for the remaining jobs from E'_k, i.e., those from some E'_i, it follows again by (ii) for (I'_i, E'_i).

Property (iii): Let j be the last job of I'_k scheduled on M_1 by GREEDY. This is the last job of I'_{k-1}, so by (iii) for I'_{k-1} GREEDY completes it at time $D(I'_{k-1}) = C_{\text{GREEDY}}(N_k)$. By (2) and (A'), we have $C_{\text{GREEDY}}(N_k) = D(I'_k)$, so GREEDY completes j at time $D(I'_k)$.

Property (iv): First consider batch e_i, $i = 2, \ldots, k$, released at time $C_{\text{ADV}}(N_i)$, when cluster N_i is busy for GREEDY. Clusters $N_{i+1}, \ldots, N_k$ are busy with their leading batches by (2) and clusters $N_{k+1}, \ldots, N_m$ are busy by the assumption (B') for I'_k. Recall that $C_{\text{ADV}}(N_i) = C_{\text{GREEDY}}(N_i) - \varepsilon = D(I'_{i-1}) - \varepsilon$. It follows from (iii) for I'_{i-1} that at time $C_{\text{ADV}}(N_i)$ when e_i is released, GREEDY is still running the last job of I'_{i-1} on M_1, hence all clusters $\mathcal{N}_m$ are busy. Now consider other jobs from E'_k; each such job is in some E'_i, thus the statement follows from (iv) for E'_i.

Property (v): We have, using the inductive assumption, $|I'_k| = 3k + \sum_{i=1}^{k-1} |I'_i| = 3k + 7 + \sum_{i=2}^{k-1} (16 \cdot 2^{i-2} - 3) = 16 \cdot 2^{k-2} - 3$ and $|E'_k| = 3k - 3 + \sum_{i=1}^{k-1} |E'_i| = 3k + \sum_{i=2}^{k-1} (9 \cdot 2^{i-2} - 3) = 9 \cdot 2^{k-2} - 3$.

The construction of I'_k is illustrated in Figures 4 and 5.

D Proof of Lemma 3.8

We use induction. Since we only consider the execution of jobs on a specific machine i, we use terms such as "contained", "completion time", and "a job is released during the execution of another job".

Consider a chain $J_1, J_2, \ldots$, where job J_ℓ has weight w_ℓ and size p_ℓ (the job interval of J_ℓ may be shorter than $\frac{p_\ell}{s_i}$ due to a preemption). For a chain with n jobs, every prefix can be seen as a

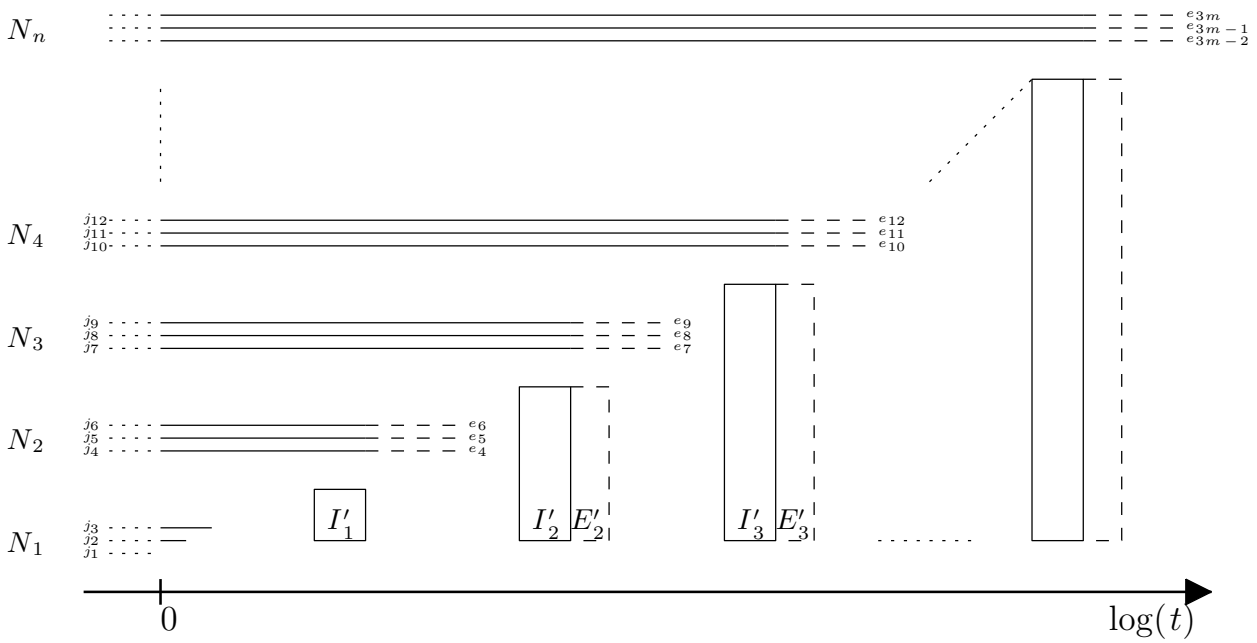

Figure 4: Overview of construction of I'_k. In this figure the ϵ is ignored, i.e., its values is set to zero. The time scale is logarithmic and starts roughly at 1, which, with ϵ set to zero, equals $C_{\text{GREEDY}}(M_1) = C_{\text{ADV}}(M_1)$. The release times of the leading batches are not depicted, since with ϵ-s set to zero, they equal zero. All extra batches are depicted.

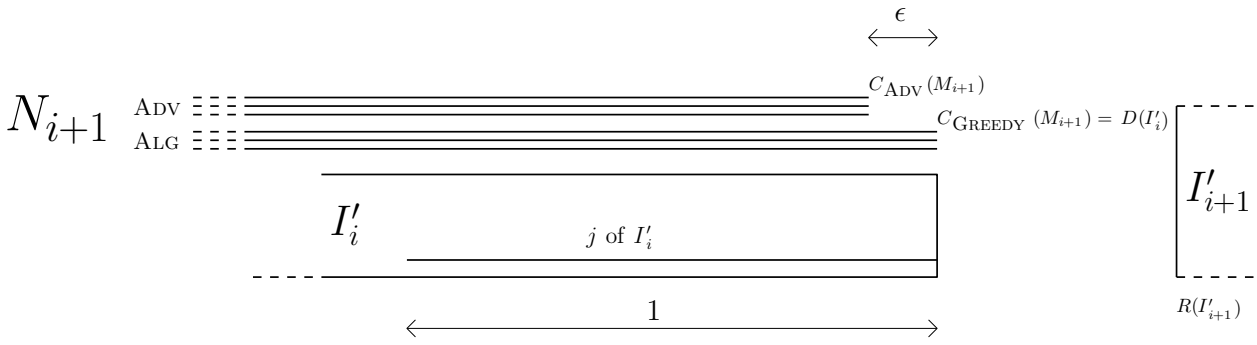

Figure 5: Details of construction of I'_k for the time around the completion(s) of the leading batches(s) on N_{i+1} by both GREEDY and ADV. In this figure the scale is roughly linear: while ϵ is smaller than 1, it might be enlarged in the figure.

chain as well, only the last job of the prefix may be preempted in the complete chain, and thus its job interval can become shorter if the entire chain is considered.

Let $\textsc{Alg}_n = \sum_{j=1}^{n} w(J_j)$ be the total profit of jobs that Alg starts during the chain consisting of n jobs. Let $\textsc{Alt}_n^1$ be the maximum possible profit of jobs of an alt-chain that are completely contained in this chain. Let $\textsc{Alt}_n$ be the maximum possible total profit of jobs of an alt-chain of the chain of n jobs.

We prove that the following two properties hold.

1. $\textsc{Alt}_n^1 \leq \textsc{Alg}_n$
2. $\textsc{Alt}_n \leq \textsc{Alg}_n + 2w_n$.

Property 2 follows immediately from Property 1 and the definition of an alt-chain: any alt-chain that is not completely contained in the chain becomes contained when we strip if of its last job. That job has weight at most $2w(J_n)$ since it contains $r(J_n)$. In what follows we will always prove the first property.

For the base case, we consider a chain consisting of a single job J_1. In an alt-chain for this chain, several jobs might be released and completed while Alg is running this job, but the total weight is maximized if the alt-chain contains only one job of length equal to the total length of all jobs of the alt-chain, due to convexity. This total length is at most p_1, so property 1 holds by monotonicity of the weight function.

We now assume the properties have been proved for chains of length n, and consider a chain of length $n+1$, and an alt-chain completely contained in it. If there is at least one job released during $[r_{n+1}, d_{n+1})$ then using convexity we can assume that there is at most one such job.

If such a job does not exist, then removing the last job of the alt-chain we get an alt-chain contained in the first n jobs since at most one job of the alt-chain can run at the time r_{n+1}. For the induction we consider the chain with complete job J_n, i.e., ending at time $r_n + \frac{p_n}{s_i}$ rather than just the job interval ending at d_n, which is smaller. The last job of the alt-chain is released while some job of the chain which precedes J_{n+1} is running, so its weight is at most $2w_n$. We get a total of $\textsc{Alg}_n + 2w_n < \textsc{Alg}_n + w_{n+1} = \textsc{Alg}_{n+1}$.

Otherwise, there is a job of the alt-chain contained in $[r_{n+1}, d_{n+1})$. Its weight is at most w_{n+1} due to monotonicity. If the other jobs in the alt-chain complete before r_{n+1} then we find a total weight of at most $\textsc{Alg}_n + w_{n+1} = \textsc{Alg}_{n+1}$ by induction. We are left with the case that there exists a job released before r_{n+1} that completes later than r_{n+1}. If this job is released before r_n then $n \geq 2$, and removing the last two jobs of the alt-chain we can use induction for $n-1$. We find a total weight of at most $\textsc{Alg}_{n-1} + 2w_{n-1} + w_{n+1} < \textsc{Alg}_{n-1} + w_n + w_{n+1} = \textsc{Alg}_{n+1}$.

Finally, we consider the case that the last two jobs of the alt-chain are such that the last one starts and completes during $[r_{n+1}, d_{n+1})$, and the previous one starts during $[r_n, d_n)$ and completes in (r_{n+1}, d_{n+1}). Let L be the total size of the two jobs, t_1 the release time of the first one, and t_2 its completion time, t_3 the release time of the second one, and t_4 its completion time. By possibly extending the second job we can assume that $t_2 = t_3$ and $t_4 = d_{n+1}$. The first job has size below p_{n+1} (since its weight is at most $2w_n$, while $f(p_{n+1}) > 2w_n$), and the second job has size below p_{n+1} since it is strictly contained in a job of such a size (its release time is strictly larger than r_{n+1}). By replacing the two jobs by a job of size p_{n+1} released at time r_{n+1} and a job of size $L - p_{n+1}$ released at time t_1, we get an alt-chain whose weight cannot be smaller than the weight of the original one by convexity. Furthermore, the last job of the modified alt-chain has weight $w_{n+1} = f(p_{n+1})$,

whereas its remainder is contained in ALG's chain without J_{n+1}. Hence, by inductive assumption, the total weight of the (modified) alt-chain is at most $\text{ALG}_n + w_{n+1} = \text{ALG}_{n+1}$.

E Details of the lower bound of m

For every long break there is a unique *critical job* which determines its length; this is the second largest of the m jobs. Precisely, if the last m jobs released before the long break are $j, \dots, j+m-1$, then the break has length $Np(j+1) - (m-2) = Np(j)/2 - m + 2 = N2^{N-j-1} - m + 2$ (and we set $r(j+m)$ to be $r(j+m-1)$ plus this last value). We show that it is indeed possible to complete all jobs until time $r(j+m)$. The adversary assigns job j to the fast machine, where it requires time $p(j) - (m-1)$ starting from the beginning of the break (time $r(j+m-1)$). Using $p(j) - (m-1) < Np(j)/2 - (m-2)$, we see that this job completes before $r(j+m)$. After this, for $k = 1, \dots, m-1$, job $j+k$ is released at time $r(j+k)$, has size $p(j)/2^k$ and after time $r(j+m-1)$ it requires time $Np(j)/2^k - (m-1-k) \leq Np(j)/2 - (m-2)$, where the inequality is easily proved using $N > 4m$. Note that we have $r(j+m) = r(j) + 1 + Np(j)/2$.

The input ends after there have been M long breaks, or if $m^2 + bm$ short breaks occur in total (in all phases together) before b long breaks have occurred. Thus the input always ends with a break. Moreover, at most N jobs will be released as claimed. This holds because between each long break and the previous break (short or long), m jobs are released, and between any short break and the previous break (short or long), at most m jobs are released, out of which the last one is assigned to a slow machine by ALG, and the previous ones are all assigned to the fast machine. Since there are at most $m^2 + Mm$ short breaks, at most $m^3 + Mm^2$ jobs are released before short breaks, for a total of $Mm + m^3 + Mm^2 = N$ jobs.

Observation E.1. *The length of short breaks and critical jobs are decreasing at least geometrically: after a short break (critical job) of length x, the next short break (critical job) has length at most $x/2$ ($x/2^m$).*

For a long break b, let t_b be the arrival time of the largest job j_b that the adversary completes during this break. The critical job of this break is then job $j_b + 1$. If the adversary does not create any long breaks, we let t_1 be the time at which the last short break (i.e., the input) finishes, and j_1 be the index of the last job that arrived plus one.

Lemma E.2. *For $b = 1, \dots, M$, the following statements hold.*

(i) The input ends before time $t_b + 2^{N-j_b+1}(N-1)$.

(ii) No job that is running on a slow machine in the schedule of ALG *can complete before the input ends.*

Proof. (i) If there are no long breaks, this holds trivially. Else, the critical job of long break b takes time $2^{N-j_b-1}N$ to process on a slow machine, so the total time used by the adversary to process all the critical jobs that are released after time t_b is at most $2^{N-j_b-1}N(1 + 2^{-m} + 2^{-2m} + \dots) = 2^{N-j_b-1}N/(1-2^{-m})$ by Observation E.1. The total length of all short breaks after time t_b is at most $2^{N-j_b-m}(1 + 1/2 + 1/2^2 + \dots) < 2^{N-j_b-m+1}$ by Observation E.1 and because the first job which is released after long break b has size exactly 2^{N-j_b-m}. At most N other jobs are released

at 1 time unit intervals. The total time that can pass after time t_b until the input ends is thus at most $\frac{2^{N-j_b-1}N}{1-2^{-m}} + 2^{N-j_b-m+1} + N$. This is less than $2^{N-j_b+1}(N-1)$ if

$$N\left(\frac{2^m}{2^m-1} + 2^{j_b+1-N}\right) + 2^{2-m} < 4N-4$$

Using $j_b \leq N$, this holds if $N(2-\frac{2^m}{2^m-1}) > 2^{2-m}+4$, which is true for $N > 4 \cdot \frac{2^m+1}{2^m} \cdot \frac{2^m-1}{2^m-2} = 4 \cdot \frac{2^{2m}-1}{2^{2m}-2^{m+1}}$. For $m \geq 2$, this last expression is at most 8, and we have $N > m^3 \geq 8$.

(ii) If $t_b = 1$, there is nothing to show: ALG does not run any job of the first phase on a slow machine. If $b > 1$ and there are no jobs between long break $b-1$ and the jobs that the adversary completes during long break b, then the claim follows by induction: no new jobs were started by ALG on slow machines after the previous long break.

In the remaining cases (including the case where there are no long breaks), job $j_b - 1$ was placed on a slow machine by ALG and caused a short break. Thus it was released at time $t_b - 2^{N-j_b+1}$ and ALG can complete it at the earliest at time $t_b - 2^{N-j_b+1} + 2^{N-j_b+1}N = t_b + 2^{N-j_b+1}(N-1)$, by which time the input has ended by (i).

If $b > 1$, then by induction, no job that was released *before* the jobs which led to long break $b-1$ can be completed by ALG on a slow machine before the input ends. We will now lower bound the completion time of the other (more recent) jobs on the slow machines (if they exist), also for the case $b = 1$. Each such job caused a short break.

We first consider a simple case, where all these jobs were released consecutively immediately before the jobs which led to long break b. In this case, the k-th such job (counting backwards from time t_b) was released at time $t_b - 2^{N-j_b+1}(1+2+\cdots+2^{k-1}) \geq t_b - k2^{N-j_b+k}$ and does not complete before time $t_b + (N-k)2^{N-j_b+k} > t_b + (N-1)2^{N-j_b+1}$, so we are again done using (i). The inequality holds because $N(2^k-2) > k2^k - 2$ which is true for all $k \geq 2, N \geq k+1$.

Note that this proves that *any* job which is released during an arbitrarily long sequence of consecutive short breaks that immediately precedes a long break can only finish (on a slow machine) after the input ends. There may also be jobs that ALG assigns to the fast machine in between. Consider all such jobs starting from the last one before time t_b. We can insert these jobs one by one into the sequence, starting from the end. The effect of each insertion is that the release date of all preceding jobs is decreased by 1 compared to the calculations above, whereas their sizes are doubled. Thus after any such job is inserted, we still have that no job which ALG is running at time t_b on a slow machine can complete before the input ends. □

Lemma E.3. ALG *cannot complete any job on the fast machine except during long breaks.*

Proof. First, consider any maximal set of consecutive jobs that ALG assigns to the fast machine. By construction, these jobs arrive at consecutive integer times, and all except maybe the very last one of the input has size more than 1. This shows that ALG could only possibly complete the last job of each such set. If the set has size m, this happens during the long break that follows. This can be seen as follows. Consider long break b. The adversary completes job $j_b + 1$ which has size 2^{N-j_b-1} on a slow machine during this break. The job that ALG assigned to the fast machine when break b started is job $j_b + m - 1$, which arrives at time $t_b + m - 1$ and has size 2^{N-j_b-m+1}. Since $m - 1 + 2^{N-j_b-m+1} < N2^{N-j_b}$, ALG completes it during the long break.

For a set of size less than m, at least one short break starts one time unit after the last job in the set arrives. Say this last job has size 2^{N-j}. Then the short break which follows has size

2^{N-j-1}, and by Observation E.1, the total length of all possible later short breaks is at most $2^{N-j-1}(1+1/2+\cdots+1/2^{-m^2-Mm}) < 2^{N-j}$. So the job of size 2^{N-j} cannot complete before either the input ends or another job is assigned by ALG to the fast machine. □

It follows from Lemmas E.2and E.3 that after the b-th long break, ALG has completed at most b jobs (the ones that it was running on the fast machine when each long break started), and none of the jobs that were released so far and that were assigned to slow machines can complete before the input ends.