Online Makespan Minimization With Budgeted Uncertainty

Susanne Albers[1] and Maximilian Janke[2]

[1]Department of Computer Science, Technical University of Munich, albers@in.tum.de
[2]Department of Computer Science, Technical University of Munich, janke@in.tum.de

Abstract

We study Online Makespan Minimization with uncertain job processing times. Jobs are assigned to m parallel and identical machines. Preemption is not allowed. Each job has a regular processing time while up to Γ jobs fail and require additional processing time. The goal is to minimize the makespan, the time it takes to process them all if these Γ failing jobs are chosen worst possible. This models real-world applications where acts of nature beyond control have to be accounted for.

So far Online Makespan Minimization With Budgeted Uncertainty has only been studied as an offline problem. We are first to provide a comprehensive analysis of the corresponding online problem.

We provide a lower bound of 2 for general deterministic algorithms showing that the problem is more difficult than its special case, classical Online Makespan Minimization. We further analyze Graham's Greedy strategy and show that it is precisely $\left(3 - \frac{2}{m}\right)$-competitive. This bound is tight. We finally provide a more sophisticated deterministic algorithm whose competitive ratio approaches 2.9052.

1 Introduction

Scheduling is universal in countless areas of computing, decision making and management: Machines simultaneously produce diverse specialized equipment; server hubs execute numerous types of programs at the same time; employees need to perform various tasks in parallel. Innumerous scheduling models in the literature are derived from such applications and model fuzzy real-world problems using precise mathematical language and problem specifications. This unnatural precision leads to failure when classical approaches are applied to real-world environments. Real-world settings are less clear-cut with multiple sources of uncertainty that vastly affect results. Consider acts of nature beyond control and predictability occurring rarely but regularly: For example, machines malfunction and need to be repaired; programs exhibit bugs, which require restarts and debugging; changes in working environment — such as a global pandemic — require new solutions such as working from home. Most tasks are eventually adapted to the situation and performed as efficiently as before. Still, a few remaining ones suddenly require a lot more time and effort in our daily schedules. Such errors in the expected difficulty of tasks easily render theoretical predictions void and motivated various models incorporating uncertainty in the literature.

A prolific line of research on online algorithms, [14, 15, 17, 19, 23, 26, 30] and references therein, considers explorable uncertainty. This type of uncertainty can be queried and explored by the online

algorithm. Such approaches are sensible for many practical applications but fail if uncertainty is inherently unpredictable — or unexplorable — even to the offline algorithm.

Offline approaches, dating back to the 1950s [13], propose stochastic models; more recent approaches, particularly Budgeted Uncertainty, consider worst-case scenarios, which accustom risk-averse decision makers. To date, many offline problems, Scheduling [8, 9, 38], Bin Packing [34] and Linear Optimization [6, 7] among them, have been analyzed under Budgeted Uncertainty assumptions.

Surprisingly, these studies never extended to online settings. Current online analyses measure the price of not having information regarding the future — an online algorithm is prepared for whatever may come — but are not the least skeptical about information once "obtained".

This paper studies the most basic scheduling problem of *Online Makespan Minimization* under *Budgeted Uncertainty* assumptions: Jobs have to be assigned to m identical and parallel machines. Preemption is not allowed. The goal is to minimize the time it takes to process them all, the *makespan*. Each job J_i is defined by two processing times. Its regular processing time $\tilde{p}_i$ is its time required under normal circumstances. Its additional time Δp_i has to be added in rare cases of failure for reparation, potential reduced performance or other application-specific slowdown. Since failures are the exception, it would be extremely pessimistic to assume that jobs in general take time $\tilde{p}_i + \Delta p_i$. Instead, one only accounts for at most Γ such failures. Given an assignment of jobs, we consider the maximum time required for processing these jobs in parallel if up to Γ failures occurred in total. This time is called the *uncertain makespan* or simply the *makespan*. The objective is to minimize this quantity.

For $\Gamma = 0$ the problem reduces to classical makespan minimization. Only regular processing times $\tilde{p}_i$ matter. Similarly, the 'paranoid case' $\Gamma = \infty$, where every program has a bug and every machine constantly malfunctions, yields classical makespan minimization using *worst-case processing times* $p_i = \tilde{p}_i + \Delta p_i$.

To an *online algorithm* A jobs are revealed one by one and each has to be scheduled immediately and irrevocably before the next one is revealed. This prevents A from performing optimally on arbitrary input sequences. The (uncertain) makespan of A, denoted by $A(\mathcal{J})$, is then compared to the optimum makespan $\mathrm{OPT}(\mathcal{J})$. Online algorithm A is c-competitive, $c \geq 1$, if $A(\mathcal{J})$ exceeds $\mathrm{OPT}(\mathcal{J})$ by at most a factor c on all input sequences $\mathcal{J}$. The smallest such factor is the *competitive ratio* $c = \sup_{\mathcal{J}} \frac{A(\mathcal{J})}{\mathrm{OPT}(\mathcal{J})}$. The goal is to design online algorithms achieving small competitive ratios.

Related work:

Scheduling is a fundamental problem in theoretical computer science and has been studied extensively in both offline and online variants, see e.g. [4, 16, 20, 22, 24, 31] and references therein. We thus only focus on results most relevant to our work beginning with robust scheduling. Early work studied arbitrary, mostly finite sets of scenarios, see e.g. [3, 28, 29, 32]. More modern work [8, 9, 38] has adapted to the model of Budgeted Uncertainty from [6]. In particular, Bougeret et. al. [9] habe provided a first 3-approximation for the offline version of the problem studied in this paper and a PTAS for constant Γ. This result has been recently improved by the same authors and Jansen [8]. They provide an EPTAS for general Γ and show that, assuming $P \neq NP$, the best possible approximation ratio for unrelated machines lies in the interval $[2, 3]$.

Online Makespan minimization is very thoroughly researched. We again only review results most relevant for our work. Already in the 1960s Graham [22] established that his famed Greedy strategy obtains a strong competitive ratio of precisely $2 - \frac{1}{m}$. It took nearly thirty years till a breakthrough of Galambos and Woeginger [21] sparked a fruitful line of research [1, 4, 27] leading to the currently best competitive ratio of 1.9201 from [20]. Chen et al. [10] have given an online algorithm whose competitiveness is at most $1 + \varepsilon$ times the best possible ratio but no explicit bounds on this ratio are obtained. For general m, lower bounds are given in [1, 5, 18, 35]. The currently best lower bound is 1.885 due to [35].

More recent results on Online Makespan Minimization consider semi-online settings, which equip the online algorithm with additional capabilities or information ahead of time [11, 16, 36]; different approaches weaken the adversaries ability to determine the job-order [2, 16]; yet other settings analyze more involved objective functions: particularly the model of vector scheduling [12, 25] also considers jobs that have two or even more "processing times". Unlike in our model, these "times" represent multidimensional resource requirements.

A related line of research on online algorithms considers "explorable uncertainty". In 1991, Kahan [26] has investigated the number of queries necessary to determine median and maximum of a set of "uncertain" numbers. This sparked a long line of research covering many problems, such as finding the median or general rank-k-elements [19, 23, 26, 30], caching [33] or most recently scheduling [14, 15]. The latter work, in fact, adds the cost of querying to the general cost paid by the algorithm for performing its task at hand. We refer to the survey by Erlebach and Hoffmann [17] and references therein for more results on this topic.

Modern work of Singla [37] introduces the 'price of information' into classical stochastic uncertainty settings, which results in a model highly related to the Budgeted Uncertainty model.

Our contribution:

We study online makespan minimization with Budgeted Uncertainty in depth. First, we give tight bounds on the competitive ratio of the Greedy strategy under Budgeted Uncertainty, which shows surprising parallels to the traditional result. It is precisely $\left(3 - \frac{2}{m}\right)$-competitive. This already beats the first offline approximation ratio provided in [9], while being a much simpler approach at the same time. The lower bound showing that Greedy is not better than $\left(3 - \frac{2}{m}\right)$-competitive can be chosen such that $\tilde{p}_i = 0$ for all i raising the question whether this important special case is as hard the general problem.

Next, we provide a better deterministic algorithm particularly suited for large values of m and Γ. Our algorithm adapts the proven strategy from [20] and earlier work of prioritizing schedules

exhibiting a steep load profile. These profiles are highly desirable and generally pose no problem to the online algorithm. It is then shown that difficult input sequences leading to less desirable profiles cannot be efficiently scheduled by any algorithm, including the optimum offline algorithm. This ameliorates the possibly higher makespan of the online algorithm on these difficult sequences.

Precise competitive ratios for small m and Γ, Figure 2, attest a strong performance unless Γ is extremely small and m is big. The latter setting is rather unnatural since one expects the number of errors to scale with the number of machines (and jobs). For large m and Γ the competitive ratio rapidly approaches 2.9052. In general, the algorithm outperforms the Greedy strategy.

We end with a lower bound of 2 for the competitive ratio achievable by any deterministic algorithm. The general model of Budgeted Uncertainty is therefore strictly more difficult than the classical model without uncertainty where even Graham's Greedy strategy is $\left(2 - \frac{1}{m}\right)$-competitive.

2 Problem definition

Consider any input sequence $\mathcal{J} = J_1, \dots J_n$. Job J_i is defined by a pair $(\tilde{p}_i, \Delta p_i)$ of non-negative real numbers where $\tilde{p}_i$ is the *regular processing time* of J_i, while Δp_i is its *additional time*. The time job J_i takes to be processed in the worst case is $p_i = \tilde{p}_i + \Delta p_i$, its *robust time*.

A schedule is simply a function $\sigma \colon \mathcal{J} \to \mathcal{M}$ mapping job $J \in \mathcal{J}$ to the machine $\sigma(J) = M \in \mathcal{M}$ processing it. The *regular load* of a machine $M \in \mathcal{M}$ is $\tilde{l}_M = \sum_{\sigma(J_i)=M} \tilde{p}_i$, the time it takes said machine to process all jobs in the best-case. The additional times we have to account for in the worst-case is the *additional load* Δl_M, the sum of the Γ largest additional times of jobs (or all additional times, if less than Γ jobs are scheduled on M). Formally $\Delta l_M = \max\left(\sum_{J_i \in \mathcal{J}'} \Delta p_i \mid \mathcal{J}' \subseteq \sigma^{-1}(M), |\mathcal{J}'| \leq \Gamma\right)$. Let us fix any set $\mathcal{J}'(M)$ where the maximum in the previous term is obtained. We break ties by preferring jobs J_i that came later, i.e. with larger indices i, and by choosing $\mathcal{J}'(M)$ of minimal size. We say for each job $J_i \in \mathcal{J}'(M)$ that J_i *fails in* σ. This allows us to write $\Delta l_M = \sum_{J_i \text{ fails } M} \Delta p_i$. Finally, the *robust load* l_M *of a machine* M is the maximum time machine M may require if up to Γ jobs fail: Formally $l_M = \tilde{l}_M + \Delta l_M = \sum_{J_i \in \sigma^{-1}(M)} \tilde{p}_i + \sum_{J_i \text{ fails } M} \Delta p_i$.

Given any algorithm A, which outputs the schedule σ, its *(robust) makespan* is then $A(\mathcal{J}) = \max_M l_M$. The goal is to design algorithms exhibiting low makespans. Technically, in the classical problem only Γ jobs fail in total. For the analysis a more general, equivalent version leads to a better intuition: Since we consider worst-case scenarios the problem stays equivalent if we allowed up to Γ jobs to fail per machine. Indeed, consider a scenario where jobs fail in the worst possible way and Γ jobs are allowed to fail per machine and pick a most loaded machine M. Make it so that all jobs outside M do not fail. The load of M and thus the makespan does not change, and now at most Γ jobs fail in total.

Let OPT be any (fixed) offline algorithm which on any input sequence $\mathcal{J}$ outputs an optimum schedule. By some abuse of notation we also denote the optimum makespan $\mathrm{OPT}(\mathcal{J})$ by OPT, if the corresponding sequence $\mathcal{J}$ is clear.

An Algorithm A is called an *online algorithm*, if it assigns each job J_i in $\mathcal{J} = J_1, \dots, J_n$ independent of future jobs, i.e. $J_{i+1}, \dots, J_n$. It's *competitive ratio* is then $c = \sup_{\mathcal{J}} \frac{A(\mathcal{J})}{\mathrm{OPT}(\mathcal{J})}$, the quantity we wish to minimize.

3 Graham's Greedy Strategy

In his seminal work [22] Graham analyzed the greedy strategy and showed that it is $\left(2-\frac{1}{m}\right)$-competitive. In general, the scheduling literature differentiates between pre-greedy and post-greedy strategies. The former simply choose a least loaded machine; the latter choose a machine such that the resulting load is minimal. For classical Makespan Minimization these notions coincide and all greedy strategies are identical up to permutations of machines. For our problem, the pre-greedy strategies perform significantly worse, which is why we focus on post-greedy strategies. Moreover, there is no "unique" post-greedy strategy anymore. Different post-greedy decisions can lead to significantly different schedules.

We then are going to establish the following theorem.

Theorem 1. *The competitive ratio of any post-greedy strategy is precisely* $3-\frac{2}{m}$.

In particular, we will also present a lower bound on which any greedy strategy does not perform better. This lower bound, interestingly, can be chosen such that $\tilde{p}=0$ for all jobs, which might be evidence that this case is not easier than the general case.

3.1 Upper Bound

Let us consider the case of classical makespan minimization, i.e. we assume that $\Delta p_i = 0$ for all jobs J_i. The core idea of Graham [22] was to consider the *average load* of any schedule, that is $\tilde{L} = \frac{1}{m}\sum_M \tilde{l}_M = \frac{1}{m}\sum_{J_i\in\mathcal{J}} \tilde{p}_i$. The second term shows that this average load is independent of the schedule considered. This has two important consequences. First, $\mathrm{OPT} \geq \tilde{L}$ since even the optimal schedule cannot have all machine loads below average. On the other hand, no scheduler, not even the worst one, can bring all machine loads above the average load $\tilde{L}$. Graham thus argues that the least loaded machine considered by his greedy strategy has load at most $\tilde{L} \leq \mathrm{OPT}$. Since the job placed on it cannot have processing time greater than OPT either, it thus cannot cause a load exceeding 2OPT. In other words, for classical makespan minimization the greedy strategy is 2-competitive.

In our setting core argument of Graham does not work anymore. For $\Delta p_i \neq 0$, the average load L is far from being independent of the schedule in question. In fact, it may differ by a factor of m. Before giving an example let us introduce the required notation. Consider the schedule computed by any algorithm A on input sequence $\mathcal{J}$ and let $L[A] = L[A,\mathcal{J}] = \frac{1}{m}\sum_M l_M$ denote its *average robust load*. Similarly, let $\Delta L[A] = \Delta L[A,\mathcal{J}] = \frac{1}{m}\sum_M \Delta l_M = L[A] - \tilde{L}$. Consider $m\cdot\Gamma$ (or more) jobs with processing vector $(\tilde{p}_i, \Delta p_i) = (0, 1/\Gamma)$. Let A be the strategy that always uses a machine of least load and let B be the algorithm which only uses one single machine. Then $L[A]=1$ while $L[B]=\frac{1}{m}$. The average robust load thus highly depends upon the algorithm considered. Interestingly, we can bound said average load using the optimum makespan OPT.

Lemma 2. *The average (robust) load* $L[A,\mathcal{J}]$ *of any algorithm* A *on input* $\mathcal{J} = J_1,\dots,J_n$ *is at most* $L[\mathrm{OPT},\mathcal{J}] + \left(1-\frac{1}{m}\right)\mathrm{OPT}(\mathcal{J})$.

Proof. Let us fix the sequence $\mathcal{J}$ and omit it from the notation. Let T be the set of jobs that fail A but not OPT. If T is empty, all jobs that fail A also fail OPT and thus $L[A,\mathcal{J}] \leq L[\mathrm{OPT},\mathcal{J}]$. The lemma follows. Else, consider $J_{\max}\in T$ of maximum additional processing time $\Delta p_{\max}$. Consider

the machine M which contains $J_{\max}$ in the optimum schedule. Since $J_{\max}$ does not fail OPT there are Γ different jobs assigned to M which fail OPT. Let G be the set of these jobs. We obtain

$$\begin{aligned}\Delta L[A,\mathcal{J}] - \Delta L[\mathrm{OPT},\mathcal{J}] &\leq \frac{1}{m}\sum_{J_i\in T} p_i - \frac{1}{m}\sum_{J_i\in G} p_i \\ &\leq \frac{1}{m}\cdot(|T|-|G|)\Delta p_{\max}.\end{aligned}$$

At most Γ jobs can fail any machine, therefore $|T| \leq \Gamma m$. By definition $|G| = \Gamma$. Finally, $\Delta p_{\max} \leq \frac{\mathrm{OPT}}{\Gamma}$. Indeed, all jobs in G have additional processing time at least $\Delta p_{\max}$, otherwise $J_{\max}$ would have failed in the optimum schedule. On the other hand, their total additional processing time is at most $\mathrm{OPT}(\mathcal{J})$. Thus $\Delta p_{\max} \leq \frac{\mathrm{OPT}(\mathcal{J})}{|G|} \leq \frac{\mathrm{OPT}(\mathcal{J})}{\Gamma}$. Now the previous inequality yields

$$\Delta L[A,\mathcal{J}] - \Delta L[\mathrm{OPT},\mathcal{J}] \leq \frac{\Gamma m - \Gamma}{m}\cdot\frac{\mathrm{OPT}}{\Gamma} = \left(1-\frac{1}{m}\right)\mathrm{OPT}(\mathcal{J}).$$

Recall that the average regular load $\tilde{L} = \frac{1}{m}\sum_{J_i}\tilde{p}_i$ is the same for every algorithm. Therefore $L[A,\mathcal{J}] - L[\mathrm{OPT},\mathcal{J}] = \Delta L[A,\mathcal{J}] - \Delta L[\mathrm{OPT},\mathcal{J}]$. Together with the previous inequality this implies $L[A,\mathcal{J}] \leq L[\mathrm{OPT},\mathcal{J}] + \left(1-\frac{1}{m}\right)\mathrm{OPT}(\mathcal{J})$. □

Lemma 3. *The makespan of any post-greedy strategy A on input $\mathcal{J}$ is at most $\left(3-\frac{2}{m}\right)\mathrm{OPT}(\mathcal{J})$.*

Proof. Let $\mathcal{J} = J_1,\ldots,J_n$. Using induction over n we may assume that the statement of the lemma holds right before job J_n is scheduled. By definition of a post-greedy assignment it then suffices to see that there exists a machine M whose load will not exceed $(3-2/m)\,\mathrm{OPT}(\mathcal{J})$ if we assign J_n to it.

Let $(p_n,\Delta p_n)$ be the processing vector of J_n. Consider the greedy schedule of the first $n-1$ jobs and preliminarily assign job J_n to any machine that causes it to fail, i.e. contains less than Γ jobs of processing time strictly exceeding Δp_n. If no such machine exists, schedule job J_n on an arbitrary machine. Let L be the average robust load of this schedule. We replace each job J_i by a job $\hat{J}_i$ whose regular processing time is $\hat{p}_i = \tilde{p}_i$ if the job does not fail this preliminary schedule and $\hat{p}_i = p_i = \tilde{p}_i + \Delta p_i$ else. Per definition that does not change the load of any machine and $L = \frac{1}{m}\sum_i \hat{p}_i$. Now, we remove the last job $\hat{J}_n$ from the machine it was scheduled on and assign it to a least loaded machine M (with regards to the processing times $\hat{p}_i$). After removing the job $\hat{J}_n$ the average load of the schedule is $L - \frac{1}{m}\cdot\hat{p}_n$. Hence, after assigning $\hat{J}_n$ to the least loaded machine the makespan is at most $L + \left(1-\frac{1}{m}\right)\hat{p}_n$. Now, replace the jobs $\hat{J}_i$ by their original variants J_i. This can only cause the load of M to decrease. Indeed, by our choice of preliminary assignment we had $\hat{p}_n = p_n$ unless there was no machine on which J_n could fail, so job J_n contributes at most $\hat{p}_n$ to the load of M. For other jobs that fail M it is clear that they continue to do so if we remove J_n. Thus, these jobs contribute processing time $p_i = \hat{p}_i$. Jobs that do not fail only contribute processing time $\tilde{p}_i \leq \hat{p}_i$.

We have shown that assigning J_n to machine M causes its load to be at most $L + \left(1-\frac{1}{m}\right)\tilde{p}_n \leq \tilde{L}[\mathcal{J},\mathrm{OPT}] + \left(1-\frac{1}{m}\right)\mathrm{OPT} + \left(1-\frac{1}{m}\right)\tilde{p}_n \leq \left(3-\frac{2}{m}\right)\mathrm{OPT}$. The first inequality is due to Lemma 2 (there is some algorithm computing the preliminary schedule), the second due to the fact that the average load $\tilde{L}[\mathcal{J},\mathrm{OPT}]$ of the optimum schedule as well as $\tilde{p}_n$, the robust processing time of any job, are both lower bounds for OPT. By definition a post-greedy strategy will not cause a makespan exceeding the one it could obtain by chosing M. □

3.2 Lower Bound

Lemma 4. *No greedy strategy can be better than $\left(3 - \frac{2}{m}\right)$-competitive, even when all jobs have regular processing time $\tilde{p} = 0$.*

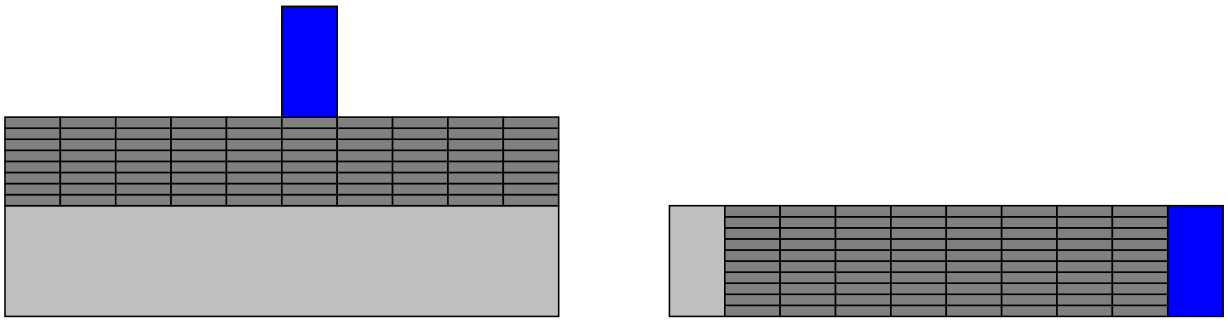

Figure 1: The lower bound for greedy strategies. First, tiny sand-like jobs fill the greedy schedule (on the left) to a height of almost 1. Small jobs increase the height to $2 - 2/m$. Finally, a large job of size 1 causes a makespan of size $3 - 2/m$. The optimum schedule (on the right) places the sand-like jobs on a single machine. Since only Γ fail the load is 1. The next $m - 1$ machines are filled with small jobs to a height of 1. The final machine captures the large job.

Proof. Let $\Gamma = m - 1 + N$ for some $N \gg 0$. Consider the following sequence in the debugging model. First, $N \cdot m$ tiny jobs of processing vector $(p, \Delta p) = (0, \Gamma^{-1})$ arrive. Then, $m(m-2)$ small jobs of processing vector $(p, \Delta p) = (0, \frac{1}{m})$ follow. Finally, one large job of processing vector $(p, \Delta p) = (0, 1)$ has to be scheduled. The greedy strategy will first assign precisely N tiny jobs to each machine, then $m - 2$ small jobs per machine and finally the large job will arrive. The machine containing the large job then contains precisely $N + m - 2 + 1 = \Gamma$ jobs. All additional processing times count! The robust load is $N \cdot \Gamma^{-1} + (m-2) \cdot \frac{1}{m} + 1 = 3 - \frac{2}{m} - \frac{m-1}{N} \xrightarrow{N \to \infty} 3 - \frac{2}{m}$.

We are left to show that OPT can schedule these jobs having makespan 1. Indeed, the first machine receives all tiny jobs. Its robust load is $\Gamma \cdot \Gamma^{-1} = 1$ since at most Γ jobs count towards the load. The next $m - 2$ machines receive m small jobs of processing time $\frac{1}{m}$ each; their load is $m \cdot \frac{1}{m} = 1$. The last remaining machine receives the single large job and thus also has load 1. □

4 An improved deterministic algorithm

The shortcoming of the Greedy strategy is that it creates 'critically flat' schedules. Jobs with high additional time tend to be spread thin onto separate machines when it would be better to cluster them. Moreover, a single large job J assigned to a flat schedule easily causes a high makespan. OPT commonly can 'sink J down' profiting a lot. Our algorithm thus tries to avoid flat schedules. When presented with one, it prefers to use a medium machine to make it steeper. Of course, recklessly using a medium machine on a dangerous schedule is folly. Said machine is only used if the algorithm can guarantee c-competitiveness. We specify the competitive ratio $c = c_{\Gamma,m}$, depending on both Γ and m, later.

For any input sequence $\mathcal{J} = J_1, \ldots, J_n$ and any time t consider the schedule of the algorithm right after after job J_t is scheduled. For $t = 0$ consider the empty schedule. We order machines by their

robust loads, breaking ties arbitrarily but consistently. We call the $d = \lfloor \frac{c-2}{c} m \rfloor \approx 0.3116m$ least loaded machines *small*; the following d machines are *medium* and the remaining $m - 2d$ most-loaded ones are *large*. Let $\mathcal{M}^t_{\text{med}}$ be the set of medium machines and let $L^t_{\text{med}} = \frac{1}{|\mathcal{M}^t_{\text{med}}|} \sum_{M \in \mathcal{M}^t_{\text{med}}} l^t_M$ be their average robust load. Let M^t_{med} be the machine in $\mathcal{M}^t_{\text{med}}$ of least robust load $l^t_{\text{med}} = \min_{M \in \mathcal{M}^t_{\text{med}}} l^t_M$. Note that $l^t_{\text{med}} \leq L^t_{\text{med}}$. We use similar notation for the small and large machines: $\mathcal{M}^t_{\text{med}}$, L^t_{med}, l^t_{med}, $\mathcal{M}^t_{\text{large}}$ etc. with the index chosen accordingly. In particular, M^t_{small} denotes the least-loaded machine. Finally, let L^t denote the average (robust) load at time t.

We call the schedule at time $t \geq 0$ *steep* if $L^{t-1}_{\text{small}} \leq \left(1 - \frac{1}{2(c-1)}\right) L^{t-1}_{\text{large}}$ and *flat* else. Steep schedules are highly desirable. Our algorithm can and will always pick the least loaded machine. If the schedule is flat, our goal should be to make it steep again. Thus, first the least loaded medium machine M^{t-1}_{med} is sampled. If scheduling a job on machine M^{t-1}_{med} will not cause its load to exceed $\frac{c}{2} L^{t-1}$, i.e. $l^{t-1}_{\text{med}} + p_t \geq \frac{c}{2} L^{t-1}$, we use M^{t-1}_{med} . This guarantees c-competitiveness, see Lemma 7. Else, the least loaded machine M_{small} has to be used. Seeing that this does not break c-competitiveness is the main part of the analysis.

Algorithm 1 *How to schedule job J_t with processing time p_t.*

1: **if** $L^{t-1}_{\text{small}} \geq \left(1 - \frac{1}{2(c-1)}\right) L^{t-1}_{\text{large}}$ and $l^{t-1}_{\text{med}} + p_t \leq \frac{c}{2} L^{t-1}$ **then**
2: Schedule job J_t on any least loaded medium machine M^{t-1}_{med};
3: **else** schedule job J_t on the least loaded machine M^{t-1}_{small}.

The values of c.

Recall that $d = \lfloor \frac{c-2}{c} m \rfloor$. The competitive ratio c is chosen minimally such that $c \geq \frac{7+\sqrt{17}}{4} \approx 2.7808$ and such that the following holds:

$$\left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^d + 2\frac{\Gamma+1}{c\Gamma} \geq \frac{2}{c-1} \cdot \frac{m-1}{m}. \tag{1}$$

Unless m is chosen extremely small c will fulfill Inequality (1) with equality. For large m and Γ this value lies below 2.9052:

Lemma 5. *For $m, \Gamma \to \infty$ the competitive ratio c approaches a value slightly below* 2.9052.

Proof. The value c will be the solution of the following equation, assuming that such a solution with $c \geq \frac{7+\sqrt{17}}{4}$ exists.

$$\left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^d + 2\frac{\Gamma+1}{c\Gamma} = \frac{2}{c-1} \cdot \frac{m-1}{m}.$$

For $m \to \infty$, we can replace d by $\frac{c-2}{c}$. We replace $\left(1 + \frac{c}{2m}\right)^d$ by $e^{\frac{cd}{2m}}$ and then by $e^{\frac{c-2}{2}}$. The term $\frac{m-1}{m}$ simply approaches 1. For $\Gamma \to \infty$, $2\frac{\Gamma+1}{c\Gamma}$ becomes $\frac{2}{c}$. Using this the previous equation simplifies for $m, \Gamma \to \infty$ to

$$\left(1 - \frac{c-2}{2c(c-1)m} - \frac{2}{c}\right) e^{\frac{c-2}{2}} + \frac{2}{c} = \frac{2}{c-1}.$$

This term does not have a closed-form solution, but we can numerically approach its solution to $c \approx 2.905186$ where the last digit is rounded up. □

The following is the main result of this paper.

Theorem 6. *The algorithm is c-competitive with $c < 2.9052$ for Γ large.*

Using a suitable data-structure that maintains the values L^{t-1}_{small}, L^{t-1}_{large} and l^{t-1}_{med} the algorithm can schedule each job efficiently in time $O(\log(m+\Gamma))$.

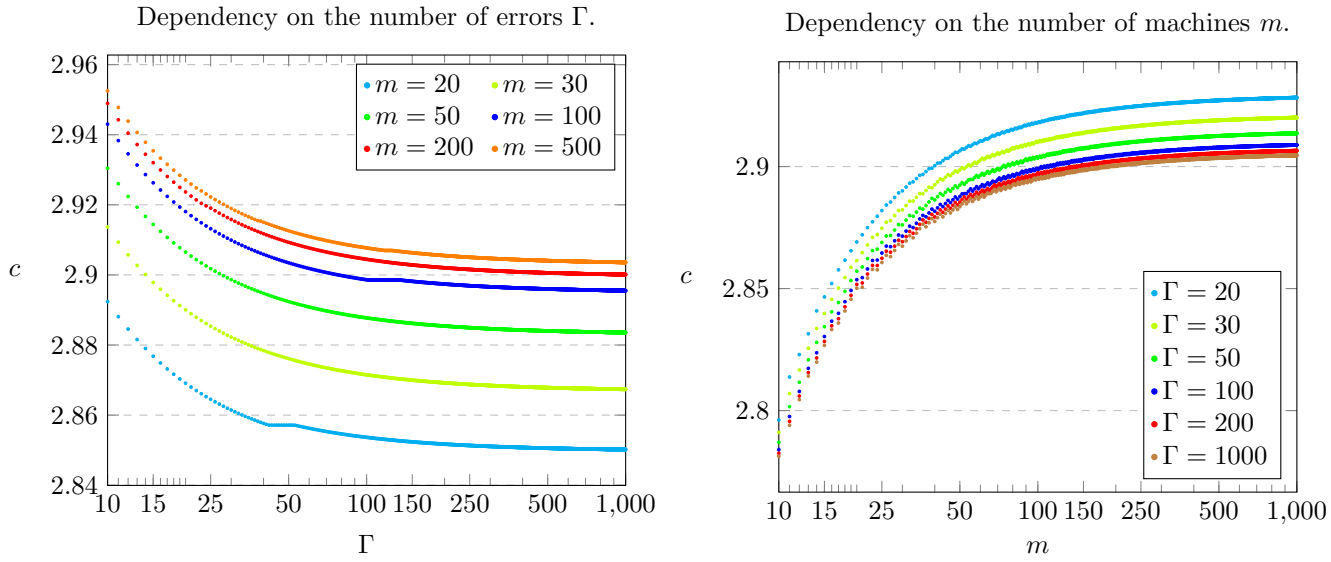

Figure 2: The competitive ratios for different m and Γ. The ratio c is monotonously decreasing in Γ and tends to increase in m albeit not monotoneously due to the rounding involved in d. The x-axes are log-scaled. The graphs are colored.

Analysis of the algorithm

Consider any input sequence $\mathcal{J} = J_1, \ldots, J_n$ and let $\text{OPT} = \text{OPT}(J_1, \ldots, J_n)$. By induction on n the makespan of the online algorithm did not exceed $c\text{OPT}$ before job J_n was scheduled. We need to show that J_n did not cause the makespan to exceed this value either. Let $L = L^n$ be the average robust load of the online schedule after all jobs in $\mathcal{J}$ have been scheduled.

Lemma 7. *We have $L \leq 2\text{OPT}$. In particular, if job J_n was scheduled on the least loaded medium machine M^{n-1}_{med} the makespan was at most $c\text{OPT}$.*

Proof. The first part follows from Lemma 2. Now, observe that if job J_n was assigned to machine M^{n-1}_{med} its load could not have exceeded $\frac{c}{2}L^{n-1} \leq c\text{OPT}$ afterwards per definition of the algorithm. □

We focus for simplicity on the case $m \to \infty$. The improvements required for small values of m are detailed at the end of the paragraph in Remark 1. Consider the case that J_n is scheduled on a least loaded machine. Let λ be its robust load. We need to bound $\lambda + p_n$, its load after receiving job J_n. Since $p_n \leq \text{OPT}$, we thus need to show that $\lambda \leq (c-1)\text{OPT}$. If $\lambda \leq \frac{c-1}{2}L$ this is a consequence of Lemma 7. Hence, we are left to consider the case where all machines have load at least $\lambda > \frac{c-1}{2}L$.

We call such a schedule *critically flat.*[1] Our algorithm cannot always prevent such schedules. The main part of our analysis shows that such schedules exhibit a highly specific structure.

Lemma 8. *If a schedule is critically flat, i.e. all machines have load $\lambda > \frac{c-1}{2}L$, then every machine contains a job of processing time at least $\frac{\lambda}{2(c-1)}$.*

We are going to prove this lemma in the next section. Let us first use it to conclude the analysis.

Proof of Theorem 6. By the previous analysis we only need to consider the case that job J_n is scheduled on a critically flat schedule where the least loaded machine has load λ. Since $\lambda \leq L \leq 2\mathrm{OPT}$ we are done if job J_n has processing time $p_n \leq \frac{\lambda}{2(c-1)} \leq \frac{\mathrm{OPT}}{c-1} \leq (c-2)\mathrm{OPT}$. The last inequality uses that $c \geq 2.7808$. If $p_n \leq \frac{\lambda}{2(c-1)}$ job J_n could only cause a makespan of $\lambda + p_n \leq 2\mathrm{OPT} + (c-2)\mathrm{OPT} = c\mathrm{OPT}$.

But else, we have shown that the sequence $J_1, \dots, J_n$ contains $m+1$ jobs of processing time $\frac{\lambda}{2(c-1)}$. One of them is J_n while the remaining m exist due to Lemma 8. By the pigeonhole principle OPT needs to place two such jobs on a single machine, attaining a makespan of at least $2\frac{\lambda}{2(c-1)} = \frac{\lambda}{c-1}$. But this shows that $\lambda \leq (c-1)\mathrm{OPT}$ and thus $\lambda + p_n \leq (c-1)\mathrm{OPT} + \mathrm{OPT} = c\mathrm{OPT}$. □

Remark 1. *Our previous definition of* critically flat *schedules given for $m \to \infty$ can be improved if m, the number of machines, is small. We then call the schedule* critically flat *if $\lambda > \frac{c-1}{2}\frac{m}{m-1}L^{n-1}$. Note that both definitions agree for $m \to \infty$. The previous arguments can be generalized to show that we are c-competitive if job J_n is not assigned to a schedule which is critically flat using this definition. A small proof is included in* **??** *for completeness.*

Proof. Let A be our algorithm then $L^{n-1} = L[A, J_1, \dots J_{n-1}]$ and set $L^n = L[A, \mathcal{J}]$. Let $\hat{p}_n$ be the processing time by which job J_n changes the total (robust) load of the machine $M = M^{n-1}_{\mathrm{small}}$ it is scheduled on. This depends on whether job J_n does fail this machine, and if it does, whether it causes another job not to fail M anymore. In general $\hat{p}_n$ could take any values in between $\tilde{p}_n$ and p_n. Using this, we see that $L^{n-1} = L^n - \hat{p}_n/m$ and the load of machine M after receiving job J_n is $\lambda + \hat{p}_n$.

Assume that $\lambda \leq \frac{c-1}{2}\frac{m}{m-1}L^{n-1}$. We have to show that $\lambda + \hat{p}_n \leq c\mathrm{OPT}$.

Using Lemma 2, we see that for our online algorithm A we have $L^n = L[A, \mathcal{J}] \leq L[\mathrm{OPT}, \mathcal{J}] + \left(1 - \frac{1}{m}\right)\mathrm{OPT}[\mathcal{J}] \leq \left(2 - \frac{1}{m}\right)\mathrm{OPT}$. In particular $L^{n-1} \leq \frac{2m-1}{m}\mathrm{OPT} - \frac{\hat{p}_n}{m}$. Using this, wee see that

$$\lambda + \hat{p}_n \leq \frac{c-1}{2}\frac{m}{m-1}L^{n-1} + \hat{p}_n \leq \frac{c-1}{2}\frac{m}{m-1}\left(\frac{2m-1}{m}\mathrm{OPT} - \frac{\hat{p}_n}{m}\right) + \hat{p}_n.$$

Recall that $\hat{p}_n \leq \tilde{p}_n \leq \mathrm{OPT}$. Using that $\frac{c-1}{2}\frac{m}{m-1}\frac{1}{m} < 1$ for $m \geq 2$ the previous term is increasing in $\hat{p}_n$ and thus maximized if we set $\hat{p}_n = \mathrm{OPT}$. Thus

$$\lambda + \hat{p}_n \leq \frac{c-1}{2}\frac{m}{m-1}\frac{2m-2}{m}\mathrm{OPT} + \mathrm{OPT} = c\mathrm{OPT}.$$ □

[1] The term 'critically flat' is not a misnomer. One can compute that such schedules cannot be steep. This is not an important observation per se but in fact a consequence of Lemma 11 as mentioned in Remark 2

Understanding how critically flat schedules are formed.

In our analysis we have to differentiate between *early* and *late* jobs. The latter will be scheduled on a full and flat schedule. For them to be assigned to a least loaded machine they will need to be fairly sizable. This requires the adversary to present quite large jobs constituting a lot of processing volume to achieve a critically flat schedule. Formally, we call a job J_t *late* if two conditions are met. We require job J_t to be scheduled on a flat schedule and we require machine M_{med}^{t-1} to have load at least λ before job J_t is scheduled. If a job J_t is not late or followed by a job which is not late, we call it *early*. In particular, a job can be both early and late. We next are going to consider late jobs.

Lemma 9. *If job J_t is late and caused a machine to first reach load λ, its (robust) processing time is at least $p_t \geq \frac{\lambda}{2(c-1)}$.*

Proof. Let $l = l_{\text{med}}^{t-1}$ be the load of M_{med}^{t-1}. By assumption of J_t being late there holds $l \geq \lambda$. Since J_t caused a machine to first reach load λ it was not scheduled on M_{med}^{t-1} but on the least loaded machine M_{small}^{t-1} instead. Since the schedule was flat, job J_t must have had processing time exceeding $\frac{c}{2}L^{t-1} - l$, i.e. $p_t > \frac{c}{2}L^{t-1} - l$. If the least loaded machine M_{small}^{t-1}, which J_t caused to reach load λ, had load less than $\lambda - \frac{\lambda}{2(c-1)}$, the statement of the lemma follows immediately. Else, all small machines had load at least $\lambda - \frac{\lambda}{2(c-1)}$ and all medium and large machines had load at least l. Thus $L^{t-1} \geq \frac{m-d}{m}l + \frac{d}{m}\left(\lambda - \frac{\lambda}{2(c-1)}\right)$. In particular

$$p_t > \frac{c}{2}L^{t-1} - l = \frac{c}{2}\left(\frac{m-d}{m}l + \frac{d}{m}\left(\lambda - \frac{\lambda}{2(c-1)}\right)\right) - l.$$

Note that d was chosen precisely maximal such that $d \leq \frac{c-2}{c}m$, or equivalently $\frac{c}{2}\frac{m-d}{m} - 1 \geq 0$. In fact, this inequality motivates the choice of d. The inequality implies that the previous term is non-decreasing in l and minimal for $l = \lambda$. Setting $l = \lambda$, we get that

$$p_t \geq \frac{c}{2}\left(1 - \frac{d}{2(c-1)m}\right)\lambda - \lambda > \left(\frac{c}{2} - \frac{c-2}{4(c-1)} - 1\right)\lambda \geq \frac{1}{2(c-1)}\lambda.$$

The first inequality uses that $d < \frac{c-2}{c}m$. The second inequality is simply an algebraic computation which uses that $c \geq \frac{7+\sqrt{17}}{4}$. □

The critical machines.

We call a time t *early* or *late* if job J_t had this property. So far, we have treated late times. Now, we need to establish a corresponding result for early times. This requires us to understand a certain set of critical machines. Given any time t, let $\mathcal{M}_{\text{crit}}^{t-1}$ be the set of d most-loaded machines whose load does not yet reached λ before job J_t is scheduled. If less than d machines fulfill the latter condition, then all of them belong to $\mathcal{M}_{\text{crit}}^{t-1}$. Let $L_{\text{crit}}^{t-1} = \frac{1}{|\mathcal{M}_{\text{crit}}^{t-1}|}\sum_{M \in \mathcal{M}_{\text{crit}}^{t-1}} l_M^{t-1}$ be their average load. We use the convention $1/0 = \infty$, or in other words set $L_{\text{crit}}^{t-1} = \infty$ if $\mathcal{M}_{\text{crit}}^{t-1} = \emptyset$. The following lemma is the central argument.

Lemma 10. *Let s be the last early time, i.e. if $t > s$, then t is late. Then $L_{\text{crit}}^{s} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$.*

We leave the proof of Lemma 10 to the next section. For now, we show how to use it to conclude the analysis.

Remark 2. *The lemma implies that $\mathcal{M}^s_{\text{crit}} \neq \emptyset$. Using that $\mathcal{M}^{n-1}_{\text{crit}} = \emptyset$ we get that $s < n-1$ or, equivalently, that critically flat schedules are always flat.*

Recall that our algorithm considers certain machines to be small, large and medium. It turns out that if it was not for the online setting, i.e. if the algorithm had advance knowledge of the sequence, the critical machines are the true contestants for the label 'medium'. To be precise we will establish that the critical machines separate 'large' machines of load at least λ from 'small' ones of load at most $\left(1 - \frac{1}{2(c-1)}\right)\lambda$. The following claim is the first step towards establishing this result in Lemma 11.

Claim 1. *Assume that $L^t_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$. If $M^{t-1}_{\text{med}} \in \mathcal{M}^{t-1}_{\text{crit}}$ then the schedule is steep. In particular, our algorithm never uses a critical machine, unless it is M^{t-1}_{small}.*

Proof. Assume $M^{t-1}_{\text{med}} \in \mathcal{M}^{t-1}_{\text{crit}}$. Since $d-1$ machines lie strictly in between M^{t-1}_{med} and the machines in $\mathcal{M}^{t-1}_{\text{large}}$ the latter must be disjoint from $\mathcal{M}^{t-1}_{\text{crit}}$. Thus, they all had load λ. In particular, $L^{t-1}_{\text{large}} \geq \lambda$. Using this, we see that the schedule was steep:

$$L^{t-1}_{\text{small}} \leq L^t_{\text{small}} \leq L^t_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda \leq \left(1 - \frac{1}{2(c-1)}\right) L^{t-1}_{\text{large}}. \qquad \square$$

Lemma 11. *If t is early $L^{t-1}_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$. Moreover job J_t was either scheduled on a machine of load at least λ or on a machine of load at most $\left(1 - \frac{1}{2(c-1)}\right)\lambda$.*

Proof. Assume for contradiction sake that an early time t existed with $L^{t-1}_{\text{crit}} > \left(1 - \frac{1}{2(c-1)}\right)\lambda$. We may wlog. choose t maximal with that property. Then there holds $L^t_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$ either by the maximality of t or by Lemma 10. Thus job J_t caused L_{crit} to decrease. This can only happen if job J_t was assigned to a machine in $\mathcal{M}^{t-1}_{\text{crit}}$ that was not M^{t-1}_{small}. But by the previous claim this does not happen, a contradiction.

For the second part observe that job J_t is either scheduled on a machine having load λ or on a machine of load at most $L^{t-1}_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$, which is either the least loaded machine in $\mathcal{M}_{\text{crit}}$ or a machine of lesser load. $\square$

We now conclude our analysis by proving the structural Lemma 8.

Proof of Lemma 8. Given any machine M, we show that job J_t that caused M to first reach (robust) load λ had processing time at least $\frac{\lambda}{2(c-1)}$. If job J_t was late, this already follows from Lemma 9. Else, by Lemma 11, job J_t was scheduled on a machine which had (robust) load $\left(1 - \frac{1}{2(c-1)}\right)\lambda$ before and λ after receiving job J_t. Thus job J_t had (robust) processing time at least $\frac{\lambda}{2(c-1)}$. $\square$

4.1 Proof of Lemma 10

We finally need to prove Lemma 10 to conclude the analysis. The behavior of our algorithm and the statement of the lemma does not change if we scale all jobs by the same constant factor. We

will thus, for ease of notation, scale all jobs such that $\lambda = 1$. Let us now assume for contradiction sake that the statement of the lemma was false, i.e. that we had

$$L^s_{\text{crit}} > 1 - \frac{1}{2(c-1)}. \tag{2}$$

Given a time $t \geq s$, let $m(t)$ denote the number of machines that have yet to reach load $\lambda = 1$ after job J_t was scheduled. Recall that $m(s) \leq d$ since by definition no job that caused the machine M_{med} to reach load $\lambda = 1$ arrived after time s. Moreover, $m(n) = m(n-1) = 0$, because all machines reached load $\lambda = 1$ before job J_n is scheduled. Given a time $t \geq s$, consider the set $\bar{\mathcal{M}}^t$ of the smallest $m(t)$ machines whose load reached at least λ. We break ties similarly to how we break ties between machine loads the same way we did before. In particular, none of large machines in $\mathcal{M}^t_{\text{large}}$ lies in $\bar{\mathcal{M}}^t$ since by assumption the d smaller machines in $\mathcal{M}^t_{\text{med}}$ have also reached load λ. In other words $\bar{\mathcal{M}}^t \subset \mathcal{M}_{\text{small}} \cup \mathcal{M}_{\text{med}}$. Let $\bar{L}^t$ be the average robust load of all machines if the load of the machines in $\bar{\mathcal{M}}_t$ is reduced to 1, that is $\bar{L}^t = \frac{1}{m}\left(\sum_{M \in \mathcal{M} \setminus \bar{\mathcal{M}}^t} l^t_M + m(t)\right)$. Note that $\bar{L}^t \leq L^t$ with equality for $t \geq n-1$. We now are going to prove the following inequality via induction for all $t \geq s$.

$$\bar{L}^t > \left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^{d-m(t)} + 2\frac{\Gamma+1}{c\Gamma}. \tag{3}$$

Note that this inequality is the motivation for our choice of c. Namely, using Inequality (1), we see that $L^{n-1} = \bar{L}^{n-1} > \frac{2}{c-1} \cdot \frac{m-1}{m}$.

Claim 2 (Base case)**.** *Inequality* (3) *holds for* $t = s$.

Proof. Observe that at time s the $m(s)$ smallest machines had average load L^s_{crit} while the next smallest $d - m(s)$ loads are at least 1. Thus

$$L_{\text{small}} \geq \frac{m(s)L^s_{\text{crit}} + (d - m(s))}{d} \geq 1 - \frac{m(s)}{2(c-1)d}.$$

The second inequality follows from Equation (2). If the schedule was steep after job J_s was scheduled we had

$$L_{\text{large}} \geq \left(1 - \frac{1}{2(c-1)}\right)^{-1} L_{\text{small}} \geq \frac{2(c-1)}{2c-1}\left(1 - \frac{m(s)}{2(c-1)d}\right).$$

If the schedule was not steep job J_s caused M_{med} to become full. Thus $m(s) = d$ and we trivially have

$$L_{\text{large}} \geq \lambda = 1 = \frac{2c-1}{2(c-1)}\left(1 - \frac{1}{2(c-1)}\right) = \frac{2c-1}{2(c-1)}\left(1 - \frac{m(s)}{2(c-1)d}\right).$$

We already argued that $\bar{\mathcal{M}}^t \cap \mathcal{M}^t_{\text{large}} = \emptyset$, thus we obtain:

$$\begin{aligned}\bar{L}^s &> \frac{(m-2d)L^s_{\text{large}} + (2d - m(s)) + m(s)L^s_{\text{crit}}}{m} \\ &\geq \frac{m-2d}{m} \cdot \frac{2c-1}{2(c-1)}\left(1 - \frac{m(s)}{2(c-1)d}\right) + \frac{2d - m(s)}{m} + \frac{m(s)}{m} \cdot \left(1 - \frac{1}{2(c-1)}\right).\end{aligned}$$

Let $f(m(s))$ denote the term on the right side of the previous inequality and let $g(m(s)) = \left(1 - \frac{d}{2(c-1)m} - \frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^{d-m(s)} + \frac{\Gamma+1}{c\Gamma}$ be the right side of Inequality 3. Then we are left to

show the following analytic fact: $f(x) \leq g(x)$ for all possible choices $x = m(s) = 0, \ldots, i$. In fact, the definition of the functions f and g canonically extends to the real numbers, so we will prove $f(x) \leq g(x)$ for all $0 \leq x \leq d$. When defined on the reals the function $f(\cdot)$ is convex, while $g(\cdot)$ is linear. Using convexity it therefore suffices to prove $f(0) \leq g(0)$ and $f(d) \leq g(d)$. One can readily compute that in the latter case $f(d) = g(d) = 1 - \frac{d}{2(c-1)m}$. By the definition of c, as mentioned right before the statement of the lemma $g(0) = \frac{1}{c-1} \cdot \frac{m+1}{m-1} < \frac{1}{c-1}$ while direct computation yields that $f(0) > \frac{1}{c-1}$ for $c > 2.52$. □

Let us treat the case $m(s) = d$. One observes that the $m(s) = d$ least loaded machines have average load at least $L_{\text{crit}} > 1 - \frac{1}{2(c-1)}$ while all other machines have load at least $\lambda = 1$. Thus

$$\begin{aligned}\bar{L}^t &> \frac{1}{m}\left((m-d) + d\left(1 - \frac{1}{2(c-1)}\right)\right) \\ &= \left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^0 + 2\frac{\Gamma+1}{c\Gamma}.\end{aligned}$$

For higher values $m(s)$, one uses the fact that the schedule is steep right before job J_s is scheduled. Using the properties of steepness one then can compute a lower bound that exceeds the right side of Inequality (3). Since this is merely computational we leave it to **??**.

Claim 3 (Inductive step)**.** *Let $s < t$. If Inequality (3) holds for $t-1$, then it holds for t.*

Proof. Indeed, consider job J_t. If J_t does not cause the load of a machine to exceed $\lambda = 1$, then this may only increase the left side of Inequality (3) and the statement of the lemma is obvious.

Else, $m(t) = m(t-1) - 1$. Let $l \geq 1$ be the load of M_{med}^{t-1} and p_t the (robust) processing time of J_t. Then $\bar{L}^t$ will increase by at least $\frac{l-1+p^t-\Gamma^{-1}}{m}$. Indeed, since the size of $\bar{\mathcal{M}}_t$ decreases by 1 a machine whose load before only counted as $\lambda = 1$ toward $\bar{L}$ will now count fully. Since its load will be at least l this causes an increase of $\frac{l-1}{m}$. Let us provisionally assume that job J_t fails machine M_{small} and that all jobs which do so, continue doing so. Then the load M_{small} will 'provisionally' increase by p_t. Of course this provisional increase may assume that $\Gamma + 1$ jobs fail machine M_{small}. If this is the case, we have to subtract Δp the smallest additional processing time of these $\Gamma + 1$ failing jobs. At least Γ jobs of additional size Δp already failed M_{small}^{t-1}. Thus $\Gamma \cdot \Delta p \leq l_{\text{small}}^{t-1} < 1$. We therefore have to subtract at most $\Delta p < \Gamma^{-1}$ for a job that ceases failing machine M_{small}^{t-1} or for job J_t not doing so. This justifies an increase in $\bar{L}$ by $\frac{p_t-\Gamma^{-1}}{m}$. Note that it does not matter if machine M_{small}^{t-1} enters $\bar{\mathcal{M}}_t$ after receiving job J_t since this will cause another machine of at least its load to leave $\bar{\mathcal{M}}_t$ in its stead. In total, we have shown that

$$\bar{L}^t \geq \bar{L}^{t-1} + \frac{1}{m}\left(l + p^t - 1 - \Gamma^{-1}\right). \tag{4}$$

Now let us consider the job J_t. Recall that since $t > s$ this job was scheduled flatly and M_{med}^{t-1} already had load $\lambda = 1$. Since it caused a machine to first reach load $\lambda = 1$ it was thus scheduled on a least loaded machine although the current schedule was flat. By definition this implies that $l + p^t > \frac{c}{2}L^{t-1} \geq \frac{c}{2}\bar{L}^{t-1}$. Combining this with Inequality (4) leads to

$$\bar{L}^t > \bar{L}^{t-1} + \frac{1}{m}\left(\frac{c}{2}\bar{L}^{t-1} - 1 - \Gamma^{-1}\right) = \left(1 + \frac{c}{2m}\right)\bar{L}^{t-1} - \frac{\Gamma+1}{m\Gamma}.$$

The claim follows by plugging the induction hypothesis for $\bar{L}^{t-1}$ into the previous term and using that by assumption $m(t) + 1 = m(t-1)$. □

The previous two claims show, using induction, that Inequality (3) holds for all $t > s$. In particular it holds for $s = n - 1$. Now combining Inequality (1) and Inequality (3) we get that

$$\bar{L}^{n-1} > \left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^{d-0} + 2\frac{\Gamma+1}{c\Gamma} \geq \frac{2}{c-1}\frac{m-1}{m}.$$

But the definition of critically flat schedules in Remark 1 implies the opposite inequality:

$$\bar{L}^{n-1} = L^{n-1} < \frac{2}{c-1}\frac{m-1}{m}\lambda = \frac{2}{c-1}\frac{m-1}{m}.$$

This is a contradiction to the assumption that Lemma 10 was not true.

4.2 Deterministic Lower Bounds

We can show that no general competitive ratio below 2 is possible unless very small numbers of machines are considered. We leave the proof to **??**.

Theorem 12. *No deterministic algorithm is better than* 2*-competitive for* $m \geq 9$ *and* $\Gamma = 2$.

Proof. Let $m \geq 9$ and $\Gamma = 2$. Consider a sequence of m *debugging jobs* $(p, \Delta p) = (0, 1)$ followed by $2(m-1)$ *real jobs* of processing vector $(p, \Delta p) = (1, 0)$ and 3 *final jobs* of processing vector $(p, \Delta p) = (3, 0)$. Consider the schedule of any (deterministic) online algorithm A after the debugging jobs are scheduled. Since so far OPT $= 1$ all debugging jobs need to be on a separate machine, else A would already cease to be 2-competitive on the prefix consisting only of debugging jobs. Now, consider the prefix consisting of all the debugging jobs and the $2(m-1)$ *real jobs*. Here, OPT $= 2$ (schedule all debugging jobs on one machine, every other machine gets two real jobs). If A is better than 2-competitive, its makespan must stay below 4 and thus it can only place 2 real jobs (in addition to the one debugging job) on any machine. By a simple counting argument at most 2 machines can have load strictly below 3 afterwards. In particular, one of the 3 final jobs needs to be placed on a machine of load 3 and thus A has a makespan of 6.

We are left to show that OPT can schedule the whole sequence having a makespan of 3. It first places all debugging jobs on one machine. Then it places all the final jobs on 3 different machines. Now, it can still schedule $3m - 11$ real jobs while maintaining a makespan of 3. In particular it can schedule $2(m-1)$ real jobs for $m \geq 9$. □

It may also be interesting to consider the imaginary case, where all jobs have real processing time 0. In this case the classical lower bounds still apply if Γ is not chosen too small. For example, the lower bound of 1.852 for [1] holds for $\Gamma \geq 4$. On the other hand it is not clear that any algorithm performs better in the case. Lemma 4 shows that Greedy does not.

References

[1] Susanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999. Publisher: SIAM.

[2] Susanne Albers and Maximilian Janke. Scheduling in the Random-Order Model. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2020.

[3] Mohamed Ali Aloulou and Federico Della Croce. Complexity of single machine scheduling problems under scenario-based uncertainty. *Operations Research Letters*, 36(3):338–342, 2008. Publisher: Elsevier.

[4] Yair Bartal, Amos Fiat, Howard Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 51–58, 1992.

[5] Yair Bartal, Howard J. Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994.

[6] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical programming*, 98(1-3):49–71, 2003. Publisher: Springer.

[7] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004. Publisher: Informs.

[8] Marin Bougeret, Klaus Jansen, Michael Poss, and Lars Rohwedder. Approximation results for makespan minimization with budgeted uncertainty. In *International Workshop on Approximation and Online Algorithms*, pages 60–71. Springer, 2019.

[9] Marin Bougeret, Artur Alves Pessoa, and Michael Poss. Robust scheduling with budgeted uncertainty. *Discrete Applied Mathematics*, 261:93–107, 2019.

[10] Lin Chen, Deshi Ye, and Guochuan Zhang. Approximating the optimal algorithm for online scheduling problems via dynamic programming. *Asia-Pacific Journal of Operational Research*, 32(01):1540011, 2015. Publisher: World Scientific.

[11] TC Edwin Cheng, Hans Kellerer, and Vladimir Kotov. Semi-on-line multiprocessor scheduling with given total processing time. *Theoretical computer science*, 337(1-3):134–146, 2005. Publisher: Elsevier.

[12] Ilan Cohen, Sungjin Im, and Debmalya Panigrahi. Online Two-Dimensional Load Balancing. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[13] George B Dantzig. Linear programming under uncertainty. *Management science*, 1(3-4):197–206, 1955. Publisher: INFORMS.

[14] Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. Scheduling with explorable uncertainty. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[15] Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. An Adversarial Model for Scheduling with Testing. *Algorithmica*, pages 1–46, 2020. Publisher: Springer.

[16] Matthias Englert, Deniz Özmen, and Matthias Westermann. The power of reordering for online minimum makespan scheduling. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 603–612. IEEE, 2008.

[17] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theoretical Computer Science*, 613:51–64, 2016. Publisher: Elsevier.

[18] Ulrich Faigle, Walter Kern, and György Turán. On the performance of on-line algorithms for partition problems. *Acta cybernetica*, 9(2):107–119, 1989.

[19] Tomas Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. Computing the median with uncertainty. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 602–607, 2000.

[20] Rudolf Fleischer and Michaela Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000. Publisher: Wiley Online Library.

[21] Gábor Galambos and Gerhard J Woeginger. An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993. Publisher: SIAM.

[22] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966. Publisher: Wiley Online Library.

[23] Manoj Gupta, Yogish Sabharwal, and Sandeep Sen. The update complexity of selection and related problems. *arXiv preprint arXiv:1108.5525*, 2011.

[24] Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987. Publisher: ACM New York, NY, USA.

[25] Sungjin Im, Nathaniel Kell, Janardhan Kulkarni, and Debmalya Panigrahi. Tight bounds for online vector scheduling. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 525–544. IEEE, 2015.

[26] Simon Kahan. A model for data in motion. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 265–277, 1991.

[27] David R Karger, Steven J Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996. Publisher: Elsevier.

[28] Adam Kasperski, Adam Kurpisz, and Pawe\l Zieliński. Approximating a two-machine flow shop scheduling under discrete scenario uncertainty. *European Journal of Operational Research*, 217(1):36–43, 2012. Publisher: Elsevier.

[29] Adam Kasperski, Adam Kurpisz, and Pawe\l Zieliński. Parallel machine scheduling under uncertainty. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 74–83. Springer, 2012.

[30] Sanjeev Khanna and Wang-Chiew Tan. On computing functions with uncertainty. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 171–182, 2001.

[31] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990. Publisher: Springer.

[32] Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. Approximating single machine scheduling with scenarios. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 153–164. Springer, 2008.

[33] Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. Technical report, Stanford, 2000.

[34] Aniket Basu Roy, Marin Bougeret, Noam Goldberg, and Michael Poss. Approximating robust bin packing with budgeted uncertainty. In *Workshop on Algorithms and Data Structures*, pages 71–84. Springer, 2019.

[35] John F. Rudin. Improved bounds for the on-line scheduling problem. 2001.

[36] Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009. Publisher: INFORMS.

[37] Sahil Singla. The price of information in combinatorial optimization. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2523–2532. SIAM, 2018.

[38] Bita Tadayon and J Cole Smith. Algorithms and complexity analysis for robust single-machine scheduling problems. *Journal of Scheduling*, 18(6):575–592, 2015. Publisher: Springer.