Online Makespan Minimization: The Power of Restart

Zhiyi Huang[1]
Department of Computer Sicence, The University of Hong Kong, Hong Kong
zhiyi@cs.hku.hk

Ning Kang
Department of Computer Sicence, The University of Hong Kong, Hong Kong
nkang@cs.hku.hk

Zhihao Gavin Tang
Department of Computer Sicence, The University of Hong Kong, Hong Kong
zhtang@cs.hku.hk

Xiaowei Wu[2]
Department of Computing, The Hong Kong Polytechnic University, Hong Kong
wxw0711@gmail.com

Yuhao Zhang
Department of Computer Sicence, The University of Hong Kong, Hong Kong
yhzhang2@cs.hku.hk

Abstract

We consider the online makespan minimization problem on identical machines. Chen and Vestjens (ORL 1997) show that the largest processing time first (LPT) algorithm is 1.5-competitive. For the special case of two machines, Noga and Seiden (TCS 2001) introduce the SLEEPY algorithm that achieves a competitive ratio of $(5-\sqrt{5})/2 \approx 1.382$, matching the lower bound by Chen and Vestjens (ORL 1997). Furthermore, Noga and Seiden note that in many applications one can kill a job and restart it later, and they leave an open problem whether algorithms with restart can obtain better competitive ratios.

We resolve this long-standing open problem on the positive end. Our algorithm has a natural rule for killing a processing job: a newly-arrived job replaces the smallest processing job if 1) the new job is larger than other pending jobs, 2) the new job is much larger than the processing one, and 3) the processed portion is small relative to the size of the new job. With appropriate choice of parameters, we show that our algorithm improves the 1.5 competitive ratio for the general case, and the 1.382 competitive ratio for the two-machine case.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms, Theory of computation → Approximation algorithms analysis, Theory of computation → Online algorithms

Keywords and phrases Online Scheduling, Makespan Minimization, Identical Machines

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2018.14

Related Version A full version of the paper can be found at [19], `https://arxiv.org/abs/1806.02207`.

[1] Partially supported by the Hong Kong RGC under the grant HKU17202115E.
[2] Part of the work was done when the author was a postdoc at the University of Hong Kong.

1 Introduction

We study in this paper the classic online scheduling problem on identical machines. Let there be m identical machines, and a set of jobs that arrive over time. For each job j, let r_j denote its release time (arrival time), and p_j denote its processing time (size). We assume without loss of generality that all r_j's and p_j's are distinct. We seek to schedule each job on one of the m machines such that the makespan (the completion time of the job that completes last) is minimized.

We adopt the standard assumption that there is a pending pool such that jobs released but not scheduled are in the pending pool. That is, the algorithm does not need to assign a job to one of the machines at its arrival; it can decide later when a machine becomes idle. Alternatively, the *immediate-dispatching* model has also been considered in some papers (e.g., [2]). We consider the standard competitive analysis of online algorithms. An algorithm is $(1+\gamma)$-competitive if for any online sequence of jobs, the makespan of the schedule made by the algorithm is at most $(1+\gamma)$ times the minimum makespan in hindsight. Without loss of generality, (by scaling the job sizes) we assume the minimum makespan $\mathsf{OPT} = 1$ (for analysis purpose only).

Chen and Vestjens [8] consider a greedy algorithm called *largest processing time first* (LPT): whenever there is an idle machine, schedule the largest job in the pending pool. They prove that the LPT algorithm is 1.5-competitive and provide a matching lower bound (consider m jobs of size 0.5 followed by a job of size 1). They also show that no online algorithm can achieve a competitive ratio better than 1.3473. For the special case when there are only two machines, Noga and Seiden [25] introduce the SLEEPY algorithm that achieves a tight $(5-\sqrt{5})/2 \approx 1.382$ competitive ratio, due to a previous lower bound given by Chen and Vestjens [8].

The 1.382 lower bound (for two machines) holds under the assumption that whenever a job is scheduled, it must be processed all the way until its completion. However, as noted in [25], many applications allow **restart**: a job being processed can be killed (put into pending) and restarted later to make place for a newly-arrived job; a job is considered completed only if it has been continuously processed on some machine for a period of time that equals to its size. In other words, whenever a job gets killed, all previous processing of this job is wasted.

Note that the restart setting is different from the *preemptive* setting, in which the processed portion is not wasted. Noga and Seiden [25] leave the following open problem: *Is it possible to beat the* 1.382 *barrier with restart?*

In this paper, we bring an affirmative answer to this long-standing open problem.

We propose a variant of the LPT algorithm (with restart) that improves the 1.5 competitive ratio for the general case, and the 1.382 competitive ratio for the two-machine case.

Our Replacement Rule. A naïve attempt for the replacement rule would be to replace a job whenever the newly-arrived job has a larger size. However, it is easy to observe that the naïve attempt fails even on one machine: the worst case competitive ratio is 2 if we keep replacing jobs that are almost completed (with jobs of slightly larger size). Hence we should prevent a job from being replaced if a large portion has been processed. Moreover, we allow a newly-arrived job to replace a processing job only if it has a much larger size, in order to avoid a long chain of replacements. As we will show by an example in Section C, the worst case competitive ratio is 1.5 if a job of size 1 is replaced by a job of size $1+\epsilon$, which is in turn replaced by a job of size $1+2\epsilon$, etc.

We hence propose the following algorithm that applies the above rules.

LPT with Restart. As in the LPT algorithm, our algorithm schedules the largest pending job whenever there is an idle machine. The main difference is that our algorithm may kill a processing job to make place for a newly-arrived job according to the following rule. Upon the arrival of a job j, we kill a processing job k (i.e., put k into pending) and schedule j if:
1. j is the largest pending job and k is the smallest among the m processing jobs;
2. the processed portion of k is less than αp_j;
3. the size of j is more than $1+\beta$ times larger than k, i.e., $p_j > (1+\beta)p_k$,

where $0 < \alpha, \beta < \frac{1}{2}$ are parameters of the algorithm. We call such an operation a *replacement* (i.e. j replaces k).

Intuitively, the parameter α provides a bound on the total amount of wasted processing (in terms of the total processing time); while the parameter β guarantees an exponential growth in the processing time of jobs if there is a chain of replacements. With appropriate choice of parameters, we show the following results.

▶ **Theorem 1.** *LPT with Restart, with parameters $\alpha = \frac{1}{200}$ and $\beta = \sqrt{2}-1$, is $(1.5 - \frac{1}{20000})$-competitive for the Online Makespan Minimization problem with restart.*

▶ **Theorem 2.** *LPT with Restart, with parameters $\alpha = \beta = 0.2$, is 1.38-competitive for the Online Makespan Minimization problem with restart on two machines.*

There are many other natural candidate replacement rules. We list some candidate algorithms that we have considered and their counter examples in Section C.

Our Techniques. The main focus of our paper is the general case, i.e., on m machines. The analysis for the two-machine case is built on the general case by refining some of the arguments. We adopt an idea from Noga and Seiden [8] to look at the last completed job in our schedule. Intuitively, only jobs with size comparable to that of the last job matter. We develop two kinds of arguments, namely the *bin-packing argument* and the *efficiency argument*.

Assume for contrary that the algorithm has a makespan strictly larger than $1+\gamma$, where $\gamma := \frac{1}{2} - \frac{1}{20000}$, we use the bin-packing argument to give an upper bound on the size of the last completed job. Assume that the last job is large, we will find a number of large jobs that cannot be packed into m bins of size 1 (recall that we assume $\mathsf{OPT} = 1$). In other words, to schedule this set of jobs, one of the m machines must get a total workload strictly greater than 1. For example, finding $2m+1$ jobs of size strictly greater than $\frac{1}{3}$ would suffice. We refer to such a set of large jobs as an *infeasible* set of jobs.

We then develop an efficiency argument to handle the case when the last job is of small size. The central of the argument is a Leftover Lemma that upper bounds the difference of total processing done by the algorithm and by OPT. As our main technical contribution, the lemma is general enough to be applied to all schedules.

Fix any schedule (produced by some algorithm ALG) and a time t. Let $\mathcal{M}$ denote the set of machines. For each machine $M \in \mathcal{M}$, let $\mathsf{W}(M,x) \in \{0,1\}$ be the indicator function of the event that "at time x, machine M is not processing while there are pending jobs". Define $\mathsf{W}_t = \sum_{M\in\mathcal{M}} \int_0^t \mathsf{W}(M,x)dx$ to be the total waste (of processing power) before time t. We show (in Section 3) the following lemma that upper bounds the leftover workload.

▶ **Lemma 3** (Leftover Lemma). *For all time t, let Δ_t be the difference in total processing time before time t between ALG and OPT. We have $\Delta_t \le \frac{1}{4}tm + \mathsf{W}_t$.*

Observe that the total processing power (of m machines) before time t is tm. The Leftover Lemma says that compared to any schedule (produced by algorithm ALG), the *extra* processing the optimal schedule can finish before time t, is upper bounded by the

processing power wasted by the schedule (e.g., due to replacements), plus a quarter of the total processing power, which comes from the sub-optimal schedule of jobs.

Consider applying the lemma to the final schedule[3] produced by our algorithm. Since our algorithm schedules a job whenever a machine becomes idle, the waste W_t comes only from the processing (before time t) of jobs that are replaced. Thus (by our replacement) we can upper bound W_t by α fraction of the total size of jobs that replace other jobs.

We remark that the above bound on the leftover workload is tight for LPT (for which $\mathsf{W}_t = 0$). Consider m jobs of size 0.5 arriving at time 0, followed by $m/2$ jobs of size $1-\epsilon$ arriving at time ϵ. The optimal schedule uses $\frac{m}{2}$ machines to process the size $(1-\epsilon)$ jobs and $\frac{m}{2}$ machines to process the size 0.5 jobs (two per machine), finishing all jobs at time 1. LPT would schedule all the size 0.5 jobs first; all of the $\frac{m}{2}$ size $(1-\epsilon)$ jobs have half of their workload unprocessed at time 1. Therefore, the amount of leftover workload at time 1 is $\frac{m}{4}$.

Other Work. The online scheduling model with restart has been investigated in the problem of scheduling jobs on a single machine to maximize the number of jobs completed before their deadlines. Hoogeveen et al. [18] study the general case and propose a 2-competitive algorithm with restart. Subsequently, Chrobak et al. [9] consider the special case when jobs have equal lengths. They propose an improved $\frac{3}{2}$-competitive algorithm with restart for this special case, and prove that this is optimal for deterministic algorithms. However, the restart rule and its analysis in our paper do not bear any obvious connections to those in [18] and [9] due to the different objectives.

Other settings of the online makespan minimization problem have been studied in the literature. A classic setting is when all machines are identical and all jobs have release time 0, but the algorithm must immediately assign each job to one of the machines at its arrival (immediate dispatching). This is the same as online load balancing problem. Graham [17] proves that the natural greedy algorithm that assigns jobs to the machine with the smallest workload is $(2-\frac{1}{m})$-competitive in this setting, which is optimal for $m \leq 3$ (due to folklore examples). A series of research efforts have then been devoted to improving the competitive ratio when m is large (e.g., [1, 3, 22]). For $m = 4$, the best upper bound is 1.7333 [7], while the best lower bound stands at 1.7321 [21]. For m that tends to infinity, the best upper bound is 1.9201 [16], while the best lower bound is 1.880 [20]. A variant of the above setting is that a buffer is provided for temporarily storing a number of jobs; when the buffer is full, one of the jobs must be removed from the buffer and allocated to a machine (e.g., [24, 10]). Kellerer et al. [23] and Zhang [27] use algorithms with a buffer of size one to achieve an improved 4/3 competitive ratio for two machines. Englert et al. [13] characterize the best ratio achievable with a buffer of size $\Theta(m)$, where the ratio is between 4/3 and 1.4659 depending on the number of machines m. When both preemption and migration are allowed, Chen et al. [6] give a 1.58-competitive algorithm without buffer, matching the previous lower bound by Chen et al. [5]. Dósa and Epstein [11] achieve a ratio of 4/3 with a buffer of size $\Theta(m)$.

Finally, if the machines are related instead of identical, the best known algorithm is 4.311-competitive by Berman et al. [4], while the best lower bound is 2 by Epstein and Sgall [15]. When preemption is allowed, Ebenlendr et al. [12] show that the upper bound can be improved to e. For the special case of two related machines, the current best competitive ratio is 1.53 by Epstein et al. [14] without preemption, and 4/3 with preemption by Ebenlendr et al. [12] and Wen and Du [26].

[3] Since a job can be scheduled and replaced multiple times, its start time is finalized only when it is completed.

Organization. We first provide some necessary definitions in Section 2. Then we prove the most crucial structural property (Lemma 3, the Leftover Lemma) in Section 3, which essentially gives a lower bound on the efficiency of all schedules. We present the details of the bin-packing argument and efficiency argument in Section 4, where our main result Theorem 1 is proved. For space reasons, the analysis for the special case of two machines is omitted. Interested readers can refer to the full version of our paper [19]. Finally, we prove in Appendix D that no deterministic algorithm, even with restart, can get a competitive ratio better than $\sqrt{1.5} \approx 1.225$.

2 Preliminaries

Consider the online makespan minimization with m identical machines and jobs arriving over time. Recall that for each job j, r_j denotes its release time and p_j denotes its size. Let OPT and ALG be the makespan of the optimal schedule and our schedule, respectively. Recall that we assume without loss of generality that $\mathsf{OPT} = 1$ (for analysis purpose only). Hence we have $r_j + p_j \leq 1$ for all jobs j. Further, let s_j and $c_j := s_j + p_j$ denote the start and completion time of job j, respectively, in the **final** schedule produced by our online algorithm. Note that a job can be scheduled and replaced multiple times. We use $s_j(t)$ to denote the last start time of j before time t.

We use n to denote the job that completes last, i.e., we have $\mathsf{ALG} = c_n = s_n + p_n$.

We consider the time horizon as continuous, and starts from $t = 0$. W.l.o.g. (by perturbing the variables slightly), we assume that all r_i's, p_i's and s_i's are different.

▶ **Definition 4** (Processing Jobs)**.** For any $t \leq \mathsf{ALG}$, we denote by $J(t)$ the set of jobs that are being processed at time t, including the jobs that are completed or replaced at t but excluding the jobs that start at t.

Note that $J(t)$ is defined based on the schedule produced by the algorithm at time t. It is possible that jobs in $J(t)$ are replaced at or after time t.

Idle and Waste. We say that a machine is *idle* in time period (a, b), if for all $t \in (a, b)$, the machine is not processing any job according to our algorithm, and there is **no** pending job. We call time t idle if there exists at least one idle machine at time t. Whenever a job k is replaced by a job j (at r_j), we say that a *waste* is created at time r_j. The size of the waste is the portion of k that is (partially) processed before it is replaced. We can also interpret the waste as a time period on the machine. We say that the waste *comes from* k, and call j the *replacer*.

▶ **Definition 5** (Total Idle and Total Waste)**.** For any $t \in [0, 1]$, define I_t as the *total idle time* before time t, i.e., the summation of total idle time before time t on each machine. Similarly, define W_t as the *total waste* before time t in the final schedule, i.e., the total size of wastes located before time t, where if a waste crosses t, then we only count its fractional size in $[0, t]$.

3 Bounding Leftover: Idle and Waste

In this section, we prove Lemma 3, the most crucial structural property. Recall that we define W_t as the total waste located before time t. For applying the lemma to general scheduling algorithms, (recall from Section 1) W_t is defined as $\sum_{M \in \mathcal{M}} \int_0^t \mathsf{W}(M, x)dx$, the total time during which machines are not processing while there are pending jobs. It is easy to check that the proofs hold under both definitions. We first give a formal definition of the *leftover* Δ_t at time t.

▶ **Definition 6** (Leftover)**.** Consider the final schedule and a fixed optimal schedule OPT. For any $t \in [0,1]$, let Δ_t be the total processing OPT does before time t, minus the total processing our algorithm does before time t.

Since the optimal schedule can process a total processing at most $m(1-t)$ after time t, we have the following useful observation.

▶ **Observation 3.1.** *The total processing our algorithm does after time t is at most $m(1-t)+\Delta_t$.*

We call time t a *marginal idle time* if t is idle and the time immediately after t is not. We first define A_t, which is designated to be an upper bound on the total processing that could have been done before time t, i.e., the leftover workload due to sub-optimal schedule.

▶ **Definition 7** (A_t)**.** For all $t \in [0,1]$, if there is no idle time before t, then define $\mathsf{A}_t = 0$, otherwise let $t' \le t$ be the last idle time before t. Define $\mathsf{A}_t = \sum_{j \in J(t')} \min\{\delta_j, p_j\}$, where $\delta_j := |T_j| = |\{\theta \in [r_j, t'] : \text{job } j \text{ is pending at time } \theta\}|$ is the total pending time of job $j \in J(t')$ before time t'.

We show the following claim, which (roughly) says that the extra processing OPT does (compared to ALG) before time t, is not only upper bounded by total idle and waste ($\mathsf{I}_t + \mathsf{W}_t$), but also by the total size or pending time of jobs currently being processed ($\mathsf{A}_t + \mathsf{W}_t$).

▶ **Claim 3.1.** *We have $\Delta_t \le \min\{\mathsf{A}_t, \mathsf{I}_t\} + \mathsf{W}_t$ for all $t \in [0,1]$.*

Proof. First observe that we only need to prove the claim for marginal idle times, as we have $\frac{d\Delta_t}{dt} \le \frac{d\mathsf{W}_t}{dt}$ (while $\frac{d\mathsf{A}_t}{dt} = \frac{d\mathsf{I}_t}{dt} = 0$) for non-idle time t. Now suppose t is a marginal idle time.

It is easy to see that Δ_t is at most $\mathsf{I}_t + \mathsf{W}_t$, the total length of time periods before t during which the algorithm is not processing (in the final schedule). Next we show that $\Delta_t \le \mathsf{A}_t + \mathsf{W}_t$.

Let $\Delta_t(t)$, $\mathsf{A}_t(t)$ and $\mathsf{W}_t(t)$ be the corresponding variables when the algorithm is run until time t. Observe that for a job $j \in J(t)$, if it is replaced after time t, then it contributes a waste to W_t but not to $\mathsf{W}_t(t)$. Moreover, it has the same contribution to $\Delta_t - \Delta_t(t)$ and to W_t. Thus we have $\mathsf{W}_t - \mathsf{W}_t(t) = \Delta_t - \Delta_t(t)$. By definition we have $\mathsf{A}_t = \mathsf{A}_t(t)$.

Hence it suffices to show that $\Delta_t(t) \le \mathsf{A}_t + \mathsf{W}_t(t)$.

Since t is idle, there is no pending job at time t. Thus the difference in total processing at time t, i.e., Δ_t, must come from the difference (between ALG and OPT) in processing of jobs in $J(t)$ that has been completed. For each $j \in J(t)$, the extra processing OPT can possibly do on j (compared to ALG) is at most $\min\{s_j(t) - r_j, p_j\}$. Hence we have $\Delta_t(t) \le \sum_{j \in J(t)} \min\{s_j(t) - r_j, p_j\}$.

Recall by Definition 7, we have $T_j \subset [r_j, s_j(t))$ is the periods during which j is pending.

Thus at every time $t \in [r_j, s_j(t)) \setminus T_j$, j is being processed (and replaced later). Hence $\big|[r_j, s_j(t)) \setminus T_j\big|$ is at most the total wastes from j that are created before $s_j(t) < t$, which implies

$$\Delta_t(t) \le \sum_{j \in J(t)} \min\{s_j(t) - r_j, p_j\} \le \sum_{j \in J(t)} \min\{\delta_j, p_j\} + \mathsf{W}_t(t) = \mathsf{A}_t + \mathsf{W}_t(t),$$

as desired. ◀

We prove the following technical claim.

▶ **Claim 3.2.** *For any integer $k \ge 1$, given any three sequences of positive reals $\{a_i\}_{i \in [k]}$, $\{b_i\}_{i \in [k]}$ and $\{h_i\}_{i \in [k]}$ satisfying conditions*

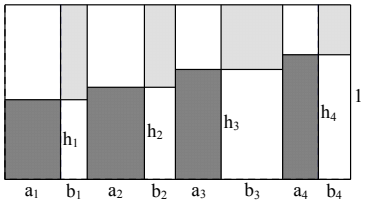

Figure 1 graph representation of Claim 3.2 for $k = 4$.

(1) $0 \le h_1 \le h_2 \le \ldots \le h_k \le 1$;
(2) *for all* $j \in [k]$, *we have* $\sum_{i\in[j]} a_i h_i \ge \frac{1}{4}\sum_{i\in[j]}(a_i + b_i)$, *we have* $\sum_{i\in[k]} b_i(1 - h_i) \le \frac{1}{4}\sum_{i\in[k]}(a_i + b_i)$.

Proof. We prove the claim by induction on k. We first show that the claim holds true when $k = 1$. Note that we have $a_1h_1 \cdot b_1(1 - h_1) \le (\frac{a_1+b_1}{2})^2 \cdot (\frac{h_1+(1-h_1)}{2})^2 = \frac{(a_1+b_1)^2}{16}$. Combine with property (2) we know that $b_1(1 - h_1) \le \frac{1}{4}(a_1 + b_1)$.

Now suppose the claim is true for all values smaller than k. Using induction hypothesis on $\{a_i\}_{i\in[k-1]}, \{b_i\}_{i\in[k-1]}$ and $\{h_i\}_{i\in[k-1]}$, we have

$$\sum_{i\in[k-1]} b_i(1 - h_i) \le \frac{1}{4}\sum_{i\in[k-1]}(a_i + b_i) \le \sum_{i\in[k-1]} a_ih_i.$$

Define $\phi = \min\{b_k, \sum_{i\in[k-1]}(4a_i \cdot h_i - a_i - b_i)\}$. Let $b'_k = b_k - \phi$ and $b'_{k-1} = b_{k-1} + \phi$.

Note that $\{a_i\}_{i\in[k]}, \{b_i\}_{i\in[k-2]} \cup \{b'_{k-1}, b'_k\}$ and $\{h_i\}_{i\in[k]}$ (and their prefixes) satisfy the conditions of the claim: first, by definition we have $b'_k > 0$ and $b'_{k-1} \ge b_{k-1} > 0$; second, since $\{a_i\}_{i\in[k]}$ and $\{h_i\}_{i\in[k]}$ are not changed, and $b'_{k-1} + b'_k = b_{k-1} + b_k$, if suffices to check condition (2) for $j = k - 1$:

$$\begin{aligned}\frac{1}{4}\sum_{i\in[k-2]}(a_i + b_i) + \frac{1}{4}(a_{k-1} + b'_{k-1}) &\le \frac{1}{4}\sum_{i\in[k-1]}(a_i + b_i) + \sum_{i\in[k-1]}(a_i \cdot h_i - \frac{a_i + b_i}{4}) \\ &= \sum_{i\in[k-1]}(a_i \cdot h_i).\end{aligned}$$

Applying the induction hypothesis on $\{a_i\}_{i\in[k-1]}, \{b_i\}_{i\in[k-2]} \cup \{b'_{k-1}\}$ and $\{h_i\}_{i\in[k-1]}$,

$$\sum_{i\in[k-1]} b_i(1 - h_i) + \phi(1 - h_{k-1}) \le \frac{1}{4}(\sum_{i\in[k-1]}(a_i + b_i) + \phi) \le \sum_{i\in[k-1]} a_ih_i.$$

If $\phi = b_k$, then immediately we have

$$\begin{aligned}\sum_{i\in[k]} b_i(1 - h_i) &\le \sum_{i\in[k-1]} b_i(1 - h_i) + b_k(1 - h_{k-1}) \\ &\le \frac{1}{4}(\sum_{i\in[k-1]}(a_i + b_i) + b_k) < \frac{1}{4}\sum_{i\in[k]}(a_i + b_i),\end{aligned}$$

as desired. Otherwise we have $\phi = \sum_{i\in[k-1]}(4a_i \cdot h_i - a_i - b_i)$, and hence we have

$$a_k \cdot h_k \ge \frac{1}{4}\sum_{i\in[k]}(a_i + b_i) - \sum_{i\in[k-1]} a_i \cdot h_i = \frac{1}{4}(a_k + b_k + \phi) = \frac{1}{4}(a_k + b'_k),$$

which implies $b'_k(1-h_k) \le \frac{1}{4}(a_k + b'_k)$. Hence we have

$$\sum_{i\in[k]} b_i(1-h_i) = \sum_{i\in[k-2]} b_i(1-h_i) + b'_{k-1}(1-h_{k-1}) + b'_k(1-h_k) + \phi(h_{k-1}-h_k)$$
$$\le \frac{1}{4}\sum_{i\in[k-2]}(a_i+b_i) + \frac{a_{k-1}+b'_{k-1}}{4} + \frac{a_k+b'_k}{4} = \frac{1}{4}\sum_{i\in[k]}(a_i+b_i),$$

which completes the induction. ◀

Given Claim 3.2, we are now ready to proof the Leftover Lemma.

Proof of Lemma 3. As before, it suffices to prove the lemma for marginal idle times, as we have $\frac{d\Delta_t}{dt} \le \frac{d\mathsf{W}_t}{dt}$ (while $\frac{d(\frac{1}{4}tm)}{dt} > 0$) for non-idle time t. Now suppose t is a marginal idle time. As before, let $\Delta_t(t)$ and $\mathsf{W}_t(t)$ be the values of variables when the algorithm is run until time t.

We prove a stronger statement that $\Delta_t(t) \le \frac{1}{4}tm + \mathsf{W}_t(t)$, by induction on the number k of marginal idle times at or before time t. Note that the stronger statement implies the lemma, as we have $\Delta_t - \Delta_t(t) = \mathsf{W}_t - \mathsf{W}_t(t)$.

In the following, we use a weaker version of Claim 3.1: we only need $\mathsf{A}_t \le \sum_{j\in J(t)} \delta_j$.

Base Case: $k = 1$. Since t is the first marginal idle time, let g be the first idle time, we know that $[g,t]$ is the only idle period. Define $J := \{j \in J(t) : s_j(t) \le g\}$ to be the set of jobs that are processed from time g to t. By definition we have $\mathsf{I}_t \le (t-g)(m-|J|)$. Recall that $\mathsf{A}_t \le \sum_{j\in J(t)} \delta_j$, where δ_j is the total pending time of job j before time t. Hence we have $\delta_j \le g$ if $j \in J$, and $\delta_j = 0$ otherwise. By Claim 3.1 we have

$$\Delta_t(t) \le \min\{\mathsf{A}_t, \mathsf{I}_t\} + \mathsf{W}_t(t) \le \min\{g|J|, (t-g)(m-|J|)\} + \mathsf{W}_t(t) \le \frac{1}{4}tm + \mathsf{W}_t(t).$$

Induction. Now suppose the statement holds for all marginal idle times $0 < t_1 < t_2 < \ldots < t_{k-1}$, and consider the next marginal idle time t_k. We show that $\Delta_{t_k}(t_k) \le \frac{1}{4}t_k m + \mathsf{W}_{t_k}(t_k)$. First of all, observe that the difference in $\Delta_{t_k}(t_k)$ and $\Delta_{t_j}(t_j)$ must come from the idle periods in $[t_j, t_k]$ and wastes created in $[t_j, t_k]$. Hence for all $j < k$ we have

$$\Delta_{t_k}(t_k) - \Delta_{t_j}(t_j) \le (\mathsf{I}_{t_k} - \mathsf{I}_{t_j}) + \mathsf{W}_{t_k}(t_k) - \mathsf{W}_{t_j}(t_j).$$

Hence, if there exists some j such that $\mathsf{I}_{t_k} - \mathsf{I}_{t_j} \le \frac{1}{4}(t_k - t_j)m$, then by induction hypothesis,

$$\Delta_{t_k}(t_k) \le \Delta_{t_j}(t_j) + \frac{1}{4}(t_k - t_j)m + \mathsf{W}_{t_k}(t_k) - \mathsf{W}_{t_j}(t_j) \le \frac{1}{4}t_k m + \mathsf{W}_{t_k}(t_k),$$

and we are done. Now suppose otherwise.

For all $i \le k$, let $g_i \in (t_{i-1}, t_i)$ be the first idle time after t_{i-1} (assume $g_0 = t_0 = 0$), i.e., $[g_1,t_1],[g_2,t_2],\ldots,[g_k,t_k]$ are the disjoint idle periods. Define $J_i := \{j \in J(t_k) : r_j \in [t_{i-1}, g_i)\}$. Note that for all $j \in J_i$, we have $\delta_j \le \sum_{x=i}^{k}(g_x - t_{x-1})$, as j is not pending during idle periods; for $j \in J(r_k) \setminus \cup_{i\le k} J_i$, we have $\delta_i = 0$. For all $i \in [k]$, define

$$a_i := t_{k-i+1} - g_{k-i+1}, b_i := g_{k-i+1} - t_{k-i}, h_i := 1 - \tfrac{1}{m}\textstyle\sum_{x\in[k-i+1]}|J_x|.$$

We show that the three sequences of positive reals $\{a_i\}_{i\in[k]}$, $\{b_i\}_{i\in[k]}$, $\{h_i\}_{i\in[k]}$ satisfy the conditions of Claim 3.2, which implies $\Delta_{t_k}(t_k) \le \mathsf{A}_{t_k} + \mathsf{W}_{t_k}(t_k) \le \frac{1}{4}t_k m + \mathsf{W}_{t_k}(t_k)$ as

$$\begin{aligned}\mathsf{A}_{t_k} &\le \sum_{j\in J(r_k)} \delta_j \le \sum_{i\in[k]} \left(\sum_{x=i}^{k}(g_x - t_{x-1})\right)|J_i| = \sum_{i\in[k]} \left((g_i - t_{i-1}) \cdot \sum_{x\in[i]} |J_x|\right) \\ &= m\sum_{i\in[k]}(b_{k-i+1}\cdot(1-h_{k-i+1})) \le \frac{1}{4}m\sum_{i\in[k]}(a_i+b_i) = \frac{1}{4}t_k m.\end{aligned}$$

Finally, we check the conditions of Claim 3.2. Condition (1) trivially holds. For condition (2), observe that $\mathsf{I}_{t_i} - \mathsf{I}_{t_{i-1}} \le (t_i - g_i)\cdot(m - \sum_{x\in[i]}|J_x|) = a_{k-i+1}\cdot h_{k-i+1}\cdot m$. Hence we have

$$\sum_{i\in[j]}(a_i\cdot h_i) \ge \frac{1}{m}\sum_{i\in[j]}(\mathsf{I}_{t_{k-i+1}} - \mathsf{I}_{k-i}) = \frac{1}{m}(\mathsf{I}_{t_k} - \mathsf{I}_{t_{k-j}}) > \frac{1}{4}(t_k - t_{k-j}) = \frac{1}{4}\sum_{i\in[j]}(a_i+b_i),$$

as required. ◀

4 Breaking 1.5 on Identical Machines

In this section, we prove Theorem 1. We will prove by contradiction: assume for contrary that $\mathsf{ALG} > 1+\gamma$, we seek to derive a contradiction, e.g., no schedule could complete all jobs before time 1 (Recall that we assume $\mathsf{OPT} = 1$). To do so, we introduce two types of arguments: we use a bin-packing argument to show that the last job must be of small size, as otherwise there exists a set of infeasible large jobs; then we use an efficiency argument (built on the Leftover Lemma) to show that the total processing (excluding idle and waste periods) our algorithm completes exceed m, the maximum possible processing OPT does.

For convenience of presentation, in the rest of the paper, we adopt the *minimum counter-example* assumption [25], i.e., we consider the instance with the minimum number of jobs such that $\mathsf{ALG} > 1+\gamma$ and $\mathsf{OPT} = 1$. As an immediate consequence of the assumption, we get that no job arrives after s_n. This is because such jobs do not affect the start time of n and therefore could be removed to obtain a smaller counter example.

Recall that in our algorithm, we set $\alpha = \frac{1}{200}$ and $\beta = \sqrt{2}-1$. Define $\gamma := \frac{1}{2} - \epsilon$, where $\epsilon = \frac{1}{20000}$. We first provide some additional structural properties of our algorithm, which will be the building blocks of our later analysis.

4.1 Structural Properties

Observe that if a job is replaced, then it must be the minimum job among the m jobs that are currently being processed. Hence immediately we have the following lemma, since otherwise we can find $m+1$ jobs (including the replacer) of size larger than $\frac{1}{2}$.

▶ **Fact 4.1** (Irreplaceable Jobs). *Any job with size at least $\frac{1}{2}$ cannot be replaced.*

Next, we show that if a job i is pending for a long time, then each of the jobs processed at time s_i must be of (relatively) large size.

▶ **Lemma 8.** *For any job i, we have $p_j > \min\{s_i - r_i, p_i\}$ for all $j \in J(s_i)$.*

Proof. It suffices to consider the non-trivial case when $s_i > r_i$. Consider any $j \in J(s_i)$. If $p_j > p_i$ or $s_j(s_i) < r_i$[4], then we have $p_j > \min\{s_i - r_i, p_i\}$. Otherwise, we consider time

[4] Note that we use $s_j(s_i)$ here instead of s_j as j can possibly be replaced after time s_i.

$s_j(s_i)$, at which job j is scheduled. Since $p_j < p_i$ and $s_j(s_i) > r_i$ (i is already released), we know that p_i must be processed at $s_j(s_i)$. Hence we know that i is replaced during $(s_j(s_i), s_i)$, which is impossible since j (which is of smaller size than i) is being processed during this period. ◀

Specifically, since $\mathsf{ALG} = s_n + p_n > 1 + \gamma$ and $r_n + p_n \leq \mathsf{OPT} = 1$, we have $s_n - r_n > \gamma$. Applying Lemma 8 to job n gives the following.

▶ **Corollary 9** (Jobs Processed at Time s_n). *We have $p_j > \min\{\gamma, p_n\}$ for all $j \in J(s_n)$.*

In the following, we show two lemmas, one showing that if a job released very early is not scheduled, then all jobs processed at that time are (relatively) large; the other showing that if a job is replaced, then the next time it is scheduled must be the completion time of a larger job.

▶ **Lemma 10** (Irreplaceable Jobs at Arrival). *If a job k is not scheduled at r_k and $r_k < \alpha p_k$, then $p_j \geq \frac{p_k}{1+\beta}$ for all $j \in J(r_k)$.*

Proof. By our replacement rule, k is not scheduled at r_k either because k is not the largest pending job at r_k, or k is the largest pending job, but the minimum job in $J(r_k)$ is not replaceable.

For the second case, since the minimum job i in $J(r_k)$ is processed at most $r_k < \alpha p_k$, job i must violate our third replacement rule, that is $p_i \geq \frac{p_k}{1+\beta}$. For the first case, let k' be the first job of size at least p_k that is not scheduled at its release time. Then we have $r_{k'} < r_k < \alpha p_k < \alpha p_{k'}$. Hence by the above argument every job in $J(r_{k'})$ is of size at least $\frac{p_k}{1+\beta} > r_k$. Thus every job in $J(r_k)$ is also of size at least $\frac{p_k}{1+\beta}$. ◀

▶ **Lemma 11** (Reschedule Rule). *Suppose some job k is replaced, then the next time k is scheduled must be the completion time of a job j such that $p_k < p_j \leq s_k$.*

Proof. Suppose k is replaced at time t and rescheduled at time t'. Since k can only replace other jobs at r_k, the next time k is scheduled must be when some machine becomes idle. And this happens only if the job j processed before t' on this machine is completed. Moreover, since k is pending from t to t', if $s_j \geq t$, i.e., k is pending when j is scheduled, then (by greedy scheduling rule) we have $p_j > p_k$; otherwise j is being processed at time t, and we also have $p_j > p_k$ as k is the smallest job among all jobs in $J(t)$ by the replacement rule. Hence, we have $s_k \geq c_j \geq p_j > p_k$. ◀

We present the central lemma for our bin-packing argument as follows. Intuitively, our bin-packing argument applies if there exists time t_1 and t_2 that are far apart, and the jobs in $J(t_1)$ and $J(t_2)$ are large (e.g. larger than $\frac{1}{3}$): if $J(t_1) \cap J(t_2) = \emptyset$, then together with job n, we have found an infeasible set of $2m + 1$ large jobs; otherwise (since t_1 and t_2 are far apart) we show that the jobs in $J(t_1) \cap J(t_2)$ must be even larger, e.g. larger than $\frac{2}{3}$.

▶ **Lemma 12** (Bin-Packing Constraints). *Given non-idle times t_1, t_2 such that $t_1 < t_2$ and $n \notin J(t_1) \cup J(t_2)$, let $a = \min_{j \in J(t_1)}\{p_j\}$ and $b = \min_{j \in J(t_2)}\{p_j\}$, none of the following cases can happen:*
(1) *$a > \frac{1}{3}$, $b > \frac{2}{3(1+\beta)}$, $t_2 - t_1 > \frac{2}{3}$ and $p_n > \frac{1}{3}$;*
(2) *$\min\{a, b, p_n\} > \frac{1}{2+\beta}$, and $t_2 - t_1 > 1 - \min\{a, b, p_n\}$.*

Proof. We show that if any of the cases happens, then we have the contradiction that $\mathsf{OPT} > 1$. We first consider case (1). We show that we can associate jobs to machines such that every machine is associated with either a job of size larger than $\frac{2}{3}$, or two jobs of size

larger than $\frac{1}{3}$. Moreover, we show that every job is associated once, while n is not associated. Note that since $p_n > \frac{1}{3}$, such an association would imply the contradiction that $\mathsf{OPT} > 1$.

First, we associate every $j \in J(t_2)$ to the machine that it is processed on. If $p_j > \frac{2}{3}$ then we are done with this machine; otherwise (when $p_j \le \frac{2}{3}$), we have $s_j(t_2) > t_1$ and we show that we can associate another job of size larger than $\frac{1}{3}$ to this machine.

- If the job $i \in J(t_1)$ processed on this machine is not replaced, or $p_i \le \frac{2}{3(1+\beta)}$, then we associate i with this machine (it is easy to check that i has not been associated before);
- otherwise the first job that completes after t_1 must be of size larger than $(1+\beta)p_i > \frac{2}{3}$, which also has not been associated before. Thus we associate it to this machine.

In both cases we are able to do the association, as claimed.

Next we consider case (2). Consider any job $i \in J(t_1)$, we have $p_i > \frac{1}{2+\beta}$. We apply an association argument similar as before: let M be the machine that job i is processed on.

- If i is replaced, then we associate the first job that completes on this machine after i is replaced, which is of size larger than $\frac{1+\beta}{2+\beta} > 1 - \min\{a, b, p_n\}$, to machine M;
- otherwise if $p_i > 1 - \min\{a, b, p_n\}$ then we associate i to M;
- otherwise we know that i completes before t_2, and we can further associate to M the job in $J(t_2)$ processed on M.

It is easy to check that every job is associated at most once. Hence every machine is associated with either a job of size larger than $1 - \min\{a, b, p_n\}$, or two jobs of size larger than $\min\{a, b, p_n\}$, which (together with $p_n > \frac{1}{2+\beta}$) gives $\mathsf{OPT} > 1$, a contradiction. ◀

4.2 Upper Bounding p_n: Bin-Packing Argument

We show in this section how to apply the structural properties from Section 4.1 to provide an upper bound $\frac{1}{2+\alpha}$ on p_n. Recall that we assume $\mathsf{ALG} > 1 + \gamma$. We show that if $p_n > \frac{1}{2+\alpha}$, then Lemma 12 leads us to a contradiction. We first prove the following lemma (which will be further used in Section 4.4), under a weaker assumption, i.e., $p_n > \frac{1}{2+\beta}$.

▶ **Lemma 13.** *If $p_n > \frac{1}{2+\beta}$, then job n is never replaced.*

Proof. Assume the contrary and consider the last time when n is replaced. Note that by Fact 4.1, we have $p_n < \frac{1}{2}$. Suppose n is replaced by job l at time r_l. Then by our replacement rule, we have $p_l \ge (1+\beta)p_n$. As $r_l + p_l \le 1$ and $s_n + p_n > 1 + \gamma$, we have

$$s_n - r_l > (1 + \gamma - p_n) - (1 - p_l) > \gamma + \beta p_n > 1 - p_n > p_n,$$

where the second last inequality holds since $\gamma > \frac{1}{2+\beta}$ and $p_n > \frac{1}{2+\beta}$. Since $r_n < r_l$, by Lemma 8, we have $\min_{j \in J(s_n)}\{p_j\} > \min\{s_n - r_n, p_n\} = p_n$. Thus we can apply Lemma 12(2), with $t_1 = r_l$, $t_2 = s_n$, $a = \min_{j \in J(t_1)}\{p_j\} = p_n$ and $b = \min_{j \in J(t_2)}\{p_j\} > p_n$, and derive a contradiction. ◀

▶ **Lemma 14** (Upper Bound on Last Job). *We have $p_n \le \frac{1}{2+\alpha}$.*

For space reasons, we defer its proof to Appendix B.

Given the upper bound $\frac{1}{2+\alpha}$ on p_n, we show the following stronger version of Lemma 13 that any job of size larger than $\frac{1}{2+\beta}$ cannot be replaced.

▶ **Corollary 15** (Irreplaceable Threshold). *Any job of size larger than $\frac{1}{2+\beta}$ cannot be replaced.*

Proof. Assume the contrary that some job j of size larger than $\frac{1}{2+\beta}$ is replaced (say, by job k at r_k). Then we have $p_k > (1+\beta)p_j$ and $\min_{i\in J(r_k)}\{p_i\} = p_j > \frac{1}{2+\beta}$. On the other hand, by Corollary 9, we have $\min_{i\in J(s_n)}\{p_i\} > \min\{p_n, \gamma\} > \frac{1}{2+\beta}$. Since $p_n < \frac{1}{2+\alpha} < \gamma$, we have

$$s_n - r_k > (1+\gamma-p_n) - (1-\frac{1+\beta}{2+\beta}) = \gamma - p_n + \frac{1+\beta}{2+\beta} > \frac{1+\beta}{2+\beta}.$$

Hence we can apply the bin-packing argument to derive a contradiction: by Lemma 12(2), with $t_1 = r_k$, $t_2 = s_n$, $a = p_j$ and $b > \frac{1}{2+\beta}$, we have a contradiction. ◀

4.3 Lower Bounding p_n: Efficiency Argument

Next we establish a lower bound on p_n, applying the Leftover Lemma. First, observe that if n is never replaced, then n is pending from r_n to s_n, where $r_n \le 1 - p_n$; if n is replaced, then n is pending from r_l to s_n, where r_l is the last time n is replaced. Since $r_l \le 1 - (1+\beta)\cdot p_n < 1 - p_n$, in both cases n is pending during time period $[1-p_n, s_n)$. Hence we have the following fact.

▶ **Fact 4.2** (Non-idle Period). *Every $t \in [1-p_n, s_n)$ is non-idle.*

As a warm-up, we show the following simple lower bound on p_n using the Leftover Lemma.

▶ **Lemma 16** (Simple Lower Bound). *We have $p_n > \frac{1}{3} - 2\alpha$.*

Proof. Let $t = 1 - p_n$. Since there is no waste after time 1, the total waste located after time t is $\mathsf{W}_1 - \mathsf{W}_t$. Since no machine is idle during time $[1-p_n, s_n)$, the total processing our algorithm does after time t is at least $m(s_n - t) - (\mathsf{W}_1 - \mathsf{W}_t)$. On the other hand, the total processing our algorithm does after time t is upper bounded by $m(1-t) + \Delta_t$ (Observation 3.1). Applying Lemma 3 (the Leftover Lemma) on Δ_t, we have

$$m(s_n - t) < m(1-t) + (\mathsf{W}_1 - \mathsf{W}_t) + \Delta_t \le m(1-t) + \mathsf{W}_1 + \frac{1}{4}tm \le m(1-t) + \alpha m + \frac{1}{4}tm.$$

Note that the last inequality follows since (by our replacement rule) each job j can only create a waste of size at most $\alpha \cdot p_j$, and the total size of jobs is at most m. Thus we have

$$\mathsf{ALG} = s_n + p_n \le 1 + \alpha + \tfrac{1-p_n}{4} + p_n = \tfrac{5}{4} + \alpha + \tfrac{3}{4}p_n,$$

which implies that (recall that we assume $\mathsf{ALG} > 1 + \gamma = \frac{3}{2} - \epsilon$)

$$p_n \ge \tfrac{4}{3}\cdot(\mathsf{ALG} - \tfrac{5}{4} - \alpha) > \tfrac{4}{3}\cdot(\tfrac{3}{2} - \epsilon - \tfrac{5}{4} - \alpha) = \tfrac{1}{3} - \tfrac{4}{3}(\epsilon + \alpha) > \tfrac{1}{3} - 2\alpha,$$

where the last inequality holds by our choices of parameter, i.e., $\epsilon = \frac{1}{20000}$ and $\alpha = \frac{1}{200}$. ◀

Observe that the Leftover Lemma provides tighter upper bounds for smaller values of t. Thus in the above proof, if we can find a smaller t such that there is no (or very little) idle time from t to s_n, then we can obtain a stronger lower bound on p_n.

▶ **Lemma 17** (Lower Bound on Last Job). *We have $p_n > \frac{1}{2+\beta}$.*

Proof. Assume for contrary that $p_n \le \frac{1}{2+\beta}$. Then by Corollary 9, we have $p_j > \min\{\gamma, p_n\} = p_n$ for all job $j \in J(s_n)$. Hence at least two jobs $k, j \in J(s_n) \cup \{n\}$ are scheduled on the same machine in OPT, such that $r_k < 1 - 2p_n$. We prove the following claim[5], which enables us to use a refined efficiency argument, i.e., apply the Leftover Lemma on a earlier time $t = r_k$. For space reasons, we omit its proof.

[5] We remark that the proof of Claim 4.1 relies on the fact that p_n is not too small. Hence Lemma 16 (the warm-up lower bound) is necessary for achieving the improved lower bound (Lemma 17).

▶ **Claim 4.1.** *The total idle time* $\mathsf{I}_{s_n} - \mathsf{I}_{r_k}$ *during time period* $[r_k, s_n]$ *is at most* $3\alpha \cdot m$.

By the above claim, the total processing ALG does after time $t = r_k$ is at least $m(s_n - t) - (\mathsf{W}_1 - \mathsf{W}_t) - 3\alpha m$. On the other hand, by Observation 3.1 and Lemma 3 we have

$$m(s_n - t) - (\mathsf{W}_1 - \mathsf{W}_t) - 3\alpha m \le m(1-t) + \Delta_t \le m(1-t) + \tfrac{1}{4}tm + \mathsf{W}_t.$$

Rearranging the inequality, we have (recall that $t = r_k < 1 - 2p_n$)

$$1 + \gamma < \mathsf{ALG} = s_n + p_n \le 1 + \tfrac{1-2p_n}{4} + 4\alpha + p_n \le \tfrac{5}{4} + 4\alpha + \tfrac{\beta}{2(2+\beta)},$$

which is a contradiction by our choice of parameters. ◀

4.4 A Hybrid Argument

We have shown that assuming $\mathsf{ALG} > 1 + \gamma$, the size of the last job n can be bounded as $\frac{1}{2+\beta} < p_n \le \frac{1}{2+\alpha}$. In the remaining part of this section, we use a hybrid argument to show that we can either use the bin-packing argument to find a set of infeasible large jobs; or derive a contradiction using the efficiency argument.

General Framework. Given that $\frac{1}{2+\beta} < p_n \le \frac{1}{2+\alpha} < \gamma$, we have $s_n = \mathsf{ALG} - p_n > 1$. Thus we have $s_j \ne r_j$ for all $j \in J(s_n)$ (as they are processed at time $s_n > 1$). Moreover, by Corollary 9, we have $\min_{j \in J(s_n)}\{p_j\} > \min\{\gamma, p_n\} > \frac{1}{2+\beta}$. In other words, there exists a set $J(s_n) \cup \{n\}$ of large jobs, each of which is never replaced (by Corollary 15), and none of them is scheduled at its release time. We know that at least two of them, say k and j (assume k is released earlier), are scheduled on the same machine in OPT. Since $s_k - r_k > (1 + \gamma - p_n - p_k) - (1 - p_k - p_j) \ge \gamma$, we know that k is pending from r_k to s_k, which is a period of length γ. Thus either our algorithm finishes a lot of processing during this period (then we can use the efficiency argument), or there are many idle and waste periods during this period (then we can use the bin-packing argument to find another m large jobs, e.g., larger than $\frac{1}{3}$) .

For the compete analysis, please refer to the full version of our paper.

References

1. Susanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999.
2. Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. *Algorithmica*, 47(3):253–268, 2007.
3. Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51(3):359–366, 1995.
4. Piotr Berman, Moses Charikar, and Marek Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35(1):108–121, 2000.
5. Bo Chen, André van Vliet, and Gerhard J. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51(5):219–222, 1994.
6. Bo Chen, André van Vliet, and Gerhard J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Operations Research Letters*, 18(3):127–131, 1995.
7. Bo Chen, André van Vliet, and Gerhard J. Woeginger. New lower and upper bounds for on-line scheduling. *Operations Research Letters*, 16(4):221–230, 1994.
8. Bo Chen and Arjen P. A. Vestjens. Scheduling on identical machines: How good is LPT in an on-line setting? *Operations Research Letters*, 21(4):165–169, 1997.

9 Marek Chrobak, Wojciech Jawor, Jirí Sgall, and Tomás Tichý. Online scheduling of equal-length jobs: Randomization and restarts help. *SIAM Journal on Computing*, 36(6):1709–1728, 2007.

10 György Dósa and Leah Epstein. Online scheduling with a buffer on related machines. *Journal of Combinatorial Optimization*, 20(2):161–179, 2010.

11 György Dósa and Leah Epstein. Preemptive online scheduling with reordering. *SIAM Journal on Discrete Mathematics*, 25(1):21–49, 2011.

12 Tomás Ebenlendr, Wojciech Jawor, and Jirí Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. *Algorithmica*, 53(4):504–522, 2009.

13 Matthias Englert, Deniz Özmen, and Matthias Westermann. The power of reordering for online minimum makespan scheduling. *SIAM Journal on Computing*, 43(3):1220–1237, 2014.

14 Leah Epstein, John Noga, Steven S. Seiden, Jirí Sgall, and Gerhard J. Woeginger. Randomized online scheduling on two uniform machines. In *SODA*, pages 317–326. ACM/SIAM, 1999.

15 Leah Epstein and Jirí Sgall. A lower bound for on-line scheduling on uniformly related machines. *Operations Research Letters*, 26(1):17–22, 2000.

16 Rudolf Fleischer and Michaela Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000.

17 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

18 Han Hoogeveen, Chris N Potts, and Gerhard J Woeginger. On-line scheduling on a single machine: maximizing the number of early jobs. *Operations Research Letters*, 27(5):193–197, 2000.

19 Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Online makespan minimization: The power of restart. *CoRR (to appear in APPROX 2018)*, abs/1806.02207, 2018.

20 J. F. Rudin III. *Improved Bound for the Online Scheduling Problem*. PhD thesis, University of Texas at Dallas, 2001.

21 J. F. Rudin III and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32(3):717–735, 2003. `arXiv:http://dx.doi.org/10.1137/S0097539702403438`.

22 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.

23 Hans Kellerer, Vladimir Kotov, Maria Grazia Speranza, and Zsolt Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997.

24 Shisheng Li, Yinghua Zhou, Guangzhong Sun, and Guoliang Chen. Study on parallel machine scheduling problem with buffer. In *IMSCCS*, pages 278–273. IEEE Computer Society, 2007.

25 John Noga and Steven S. Seiden. An optimal online algorithm for scheduling two machines with release times. *Theoretical Computer Science*, 268(1):133–143, 2001.

26 Jianjun Wen and Donglei Du. Preemptive on-line scheduling for two uniform processors. *Operations Research Letters*, 23(3-5):113–116, 1998.

27 Guochuan Zhang. A simple semi on-line algorithm for p2//c_{max} with a buffer. *Information Processing Letters*, 61(3):145–148, 1997.

A Overview of Techniques for the Two Machine Case

Our analysis for the two-machine case follows the same framework as the general case: we use a bin-packing argument to upper bound the size of last job, and use an efficiency argument to lower bound the size of last job, and finally use a hybrid argument to handle the boundary case. In this section, we will overview two technical ingredients that are specifically developed for the two-machine case, namely, a structural result that upper bounds the number of times that large jobs are replaced, and a refined efficiency argument. Similar to the general case, most of the difficulties arise when the last job has medium size. For concreteness, readers may consider the size of the last job n as slightly larger than $\gamma = 0.38$, say, $p_n = 0.4$.

Recall that for the two-machine case we fix the parameters $\beta = \alpha = 0.2$.

A.1 Bounding Major Replacements

We are interested in jobs that have size at least p_n. We call such jobs *major jobs* and refer to the replacements of major jobs as *major replacements*. In the case that there are only two machines, the number of major jobs is at most 4 by a simple bin-packing argument. (Recall that we focus on medium size last job, say $p_n = 0.4$.) Further, only major jobs can replace major jobs. Hence, we can show sharp bound on the number of major replacements. In particular, we show that when the last job is of medium size, there is either no major replacement, or at most one major replacement, depending on the size of the last job. This is formulated as the following lemmas.

▶ **Lemma 18.** *If ALG* > 1.38 *and* $p_n > \frac{1}{2+\alpha}$*, then there is no major replacement.*

Proof. Recall that for the two machines case, we set $\beta = \alpha$. First we show that job n cannot be replaced given $p_n > \frac{1}{2+\alpha}$. Observe that the replacer of n must be of size $(1+\alpha)p_n > 1 - \frac{1}{2+\alpha}$, which cannot be scheduled on the same machine with n in the optimal schedule. Suppose n is replaced (which can happen at most once) by l. Then we know that n is pending from r_l to s_n, where

$$s_n - r_l > (1.38 - p_n) - (1 - (1+\alpha)p_n) = 0.38 + \alpha p_n > \frac{1}{2+\alpha}.$$

Hence by Lemma 8, we have $p_k, p_j > \frac{1}{2+\alpha}$, where $\{k, j\} = J(s_n)$ are the two jobs processed at time s_n. If $l \notin \{k, j\}$, then none of k, j, n can be scheduled on the same machine with l in OPT, which yields a contradiction; otherwise suppose $l = k$. Then we must have $J(r_l) = \{n, j\}$ as otherwise we also have the same contradiction as just argued. Then n and j must be scheduled on the same machine in OPT. If $r_n < r_j$, then $r_n < 1 - p_n - p_j < 1 - \frac{2}{2+\alpha} < \alpha p_n$. Hence n must be scheduled at r_n, as otherwise by Lemma 10 the two jobs in $J(r_n)$ are of size larger than $\frac{p_n}{1+\alpha} > \frac{1}{3}$. Since

$$r_l - r_n > s_j - r_n > (1.38 - p_n - p_j) - (1 - p_n - p_j) = 0.38 > \alpha,$$

it is impossible for l to replace n as n has been processed a portion larger than α. If $r_j < r_n$, then for the same reasoning j is scheduled at r_j. As $s_j - r_j > (1.38 - p_n - p_j) - (1 - p_n - p_j) = 0.38$, we know that j is replaced, which is impossible as by Lemma 11, the job completed at s_j is a job of size larger than $\frac{1}{2+\alpha}$, apart from l, j and n.

Hence we know that n is never replaced. Now suppose some other job of size larger than $\frac{1}{2+\alpha}$ is replaced at time r_x. Then we have $p_x > 1 - \frac{1}{2+\alpha}$, while the two jobs in $J(r_x)$ are of size larger than $\frac{1}{2+\alpha}$. Note that since n is never replaced, it is not scheduled before s_n. Further by our assumption that no job arrives after s_n, we have $r_x < s_n$. Thus $n \notin J(r_x)$, which implies a contradiction. ◀

▶ **Lemma 19.** *If ALG* > 1.38 *and* $p_n \in (0.38, \frac{1}{2+\alpha}]$*, then there is at most one major replacement.*

The main idea is that, when there are two or more major replacements, then we can find at least four major jobs. Hence as long as we can find a job of size "not too small", then the bin-packing argument yields a contradiction.

Why is bounding the number of major replacements useful? Recall that in the efficiency argument, we upper bound the total waste by $\alpha m = 2\alpha$ using the fact that each job can only create waste once at its release time if it replaces some other job, and the amount of waste created is at most α times the size of the replacer. Suppose we can find one major job that does not replace any other job at its arrival. Then, the upper bound of the total waste will significantly decrease by αp_n. (Recall that we focus on medium size last job, say $p_n = 0.4$.)

How do we find major jobs that do not replace other jobs at their arrival? It turns out we can argue that in order to have $\mathsf{ALG} > 1 + \gamma$, there must exists some major jobs whose final start times do not equal their release times. For example, the last job n's final start time definitely does not equal its release time. For each of these jobs, if it replaces some other job at its arrival, it must be replaced later on in order to get rescheduled at its final start time. Hence, a major replacement must occur for each of these jobs. By bounding the number of major replacements, we get that some of these major jobs must not replace other jobs at their arrivals.

A.2 Refined Efficiency Argument

The second technical ingredient is a more careful efficiency argument. Let t be the last idle time before s_n. The total amount of work done in the optimal schedule after time t is at most $2(1-t)$. (Recall that $\mathsf{OPT} = 1$ and there are $m = 2$ machines.)

How much work does the algorithm process after time *t*? The algorithm is fully occupied from time t to time s_n. Further, let P be the total processing our algorithm does after s_n. Then, the total amount of processing power after time t by the algorithm is $2(s_n - t) + \mathsf{P}$. However, some of the processing power is wasted (due to replacements) and some of the workload could have been done before time t in OPT (the leftover). Recall that W_t is the amount of waste before time t. Hence, the amount of waste located after time t is $\mathsf{W}_1 - \mathsf{W}_t$. Also recall that the amount of leftover is denoted as Δ_t. So we have: (the first inequality comes from Observation 3.1)

$$2(1-t) \geq 2(s_n - t) + \mathsf{P} - \Delta_t - (\mathsf{W}_1 - \mathsf{W}_t) \geq 2(s_n - t) + \mathsf{P} - \frac{t}{2} - \mathsf{W}_1,$$

where the second inequality follows by the leftover lemma. Rearranging terms, the above implies:

$$\mathsf{ALG} = s_n + p_n \leq 1 + p_n - \frac{\mathsf{P}}{2} + \frac{t}{4} + \frac{\mathsf{W}_1}{2}.$$

Then, we will bound each of the terms P, t, and W_1: P is trivially lower bounded by p_n; t is trivially upper bounded by 1, while a more careful argument shows that $t \leq 1 - p_n$ (Fact 4.2); and W_1 is trivially upper bounded by 2α. If we plug in the trivial bounds, we get: $\mathsf{ALG} \leq 1.25 + \alpha + \frac{p_n}{2}$.

Hence, we recover the efficiency argument similar to what we have used in Section 4.1 (except that we further relax $\frac{m-1}{m} p_n \leq p_n$ in the general case). However, if we can obtain

improved bounds on P, t, or W_1, we would have a better efficiency argument for upper bounding ALG. It is usually impossible to get better bounds for all of these three quantities. Nonetheless, we manage to do so for at least one of them in all cases.

We have already provided an argument for getting a better upper bound of W_1 by bounding the number of major replacements. Next, we present some intuitions why it is possible to get improved bounds for P and t.

Since the jobs are scheduled greedily, if a replaced job x is of size smaller than p_n, then it often happens that job x is rescheduled after s_n. Hence while we suffer a loss in the total waste W_1 due to the waste that comes from x, we have an extra gain of p_x in P. In general, we will develop a unified upper bound on $\mathsf{W}_1 - \mathsf{P}$, which measures the net waste due to replacements.

Apart from the trivial upper bound $1 - p_n$ on t (by Fact 4.2), we can often derive better upper bounds on t if we do not have a good upper bound on $\mathsf{W}_1 - \mathsf{P}$. In the case when the total net waste is large, we can usually find many major jobs processed at the end of the schedule. As we have only $m = 2$ machines, during idle periods, only one job can be processed while no job is pending. Suppose we find four major jobs processed at the end of the schedule, then at least three of them must be released after the last idle time t. Since $\mathsf{OPT} = 1$, we must have $t \leq 1 - 2p_n$, which gives a better upper bound on t.

B Missing Proofs

Proof of Lemma 14. We first show a weaker upper bound: $p_n \leq \frac{1+\beta}{2}$. Assume the contrary that $p_n > \frac{1+\beta}{2} > \frac{1}{2}$. As shown in the proof of Lemma 8, for all $j \in J(s_n)$, if $s_j > r_n$, we have $p_j > p_n > \frac{1+\beta}{2} > 1-\gamma$; otherwise $s_j < r_n$ and we have $p_j > s_n - r_n \geq (1+\gamma-p_n)-(1-p_n) = \gamma$. Among the $m+1$ jobs $J(s_n)\cup\{n\}$, there exist two jobs, say k and j, that are scheduled on the same machine in OPT. Moreover, we have $s_k, s_j < r_n$, since otherwise one of them is larger than $1-\gamma$ and they cannot be completed in the same machine within makespan 1. Let k be the one with a smaller release time, i.e., $r_k < r_j$. Then we have $r_k \leq 1 - p_k - p_j < 1 - 2\gamma < \alpha p_k$.

Observe that k is never replaced, as otherwise (by Lemma 11, the reschedule rule) we have $s_k > p_k$, which implies $r_n + p_n > s_k + p_n > p_k + p_n > \gamma + \frac{1+\beta}{2} > 1$, a contradiction.

By Lemma 10, we know that $s_k = r_k$ (k is scheduled at its arrival time r_k), as otherwise the minimum job in $J(r_k)$ is of size at least $\frac{\gamma}{1+\beta}$. Thus we have $s_k \geq \frac{\gamma}{1+\beta}$, which is also a contradiction as $r_n + p_n > s_k + p_n > \frac{\gamma}{1+\beta} + \frac{1+\beta}{2} > 1$ (recall that $\gamma = \frac{1}{2} - \epsilon$ and $\beta = \sqrt{2} - 1$).

By $r_k + p_k + p_j \leq 1$ and $r_k + p_k + p_n > 1 + \gamma$, we have $p_n > 2\gamma$. Hence $r_n < 1 - 2\gamma < \alpha p_n$. By Lemma 10, we know that the minimum job in $J(r_n)$ is of size at least $\frac{p_n}{1+\beta} > \frac{1}{2}$. Hence we have the contradiction that there are $m+1$ jobs, namely $J(r_n)\cup\{n\}$, of size larger than $\frac{1}{2}$.

Hence we have that $p_n \leq \frac{1+\beta}{2}$. Now assume that $p_n > \frac{1}{2+\alpha}$.

By Lemma 13, we know that n is never replaced. By Corollary 9, all jobs in $J(s_n)\cup\{n\}$ are of size at least $\min\{\gamma, p_n\} \geq \frac{1}{2+\alpha}$. Let $k, j \in J(s_n)\cup\{n\}$ be scheduled on the same machine in OPT such that $r_k \leq 1 - p_k - p_j < 1 - \frac{2}{2+\alpha} \leq \alpha p_k$. Note that different from the previous analysis (when $p_n > \frac{1+\beta}{2}$), it is possible that $n \in \{k, j\}$.

By Lemma 10, if k is not scheduled at r_k, then each job $i \in J(r_k)$ has size $p_i \geq \frac{p_k}{1+\beta} \geq \frac{1}{(2+\alpha)\cdot(1+\beta)} > \frac{1}{3}$. Then we can apply Lemma 12(1) with $t_1 = r_k$ and $t_2 = s_n$ to derive a contradiction (observe that $t_2 - t_1 = s_n - r_k > (1+\gamma-p_n) - (1-p_k-p_j) > \frac{3}{2+\alpha} - \frac{1+\beta}{2} \geq \frac{2}{3}$).

Hence we know that k is scheduled at time r_k (thus we conclude that $k \neq n$).

We show that k must be replaced (say, by job l at time r_l) , as otherwise by $r_k + p_k + p_n > 1 + \gamma$ and $r_k + p_k + p_j \leq 1$, we have $p_n > \gamma + p_j > \gamma + \frac{1}{2+\alpha} > \frac{1+\beta}{2}$, which is a contradiction.

By Lemma 11, we have $s_k > p_k \geq \frac{1}{2+\alpha}$. We also have that $p_n \leq 1 - \frac{1}{2+\alpha}$, as otherwise $r_n \leq 1 - p_n < \frac{1}{2+\alpha} < s_k$ (k is scheduled at s_k, when n is pending), which implies $p_k > p_n > 1 - \frac{1}{2+\alpha} > \frac{1}{2}$, contradicting Fact 4.1 (jobs larger than $\frac{1}{2}$ cannot be replaced). Since $p_l > (1+\beta)p_k$, we have

$$s_n - r_l > (1+\gamma-p_n) - (1-(1+\beta)p_k) = (1+\beta)p_k + \gamma - p_n > \frac{1+\beta}{2+\alpha} + \gamma - \frac{1+\alpha}{2+\alpha} > \frac{1+\alpha}{2+\alpha}.$$

Then we can apply Lemma 12(2), with $t_1 = r_l$, $t_2 = s_n$, $a = p_k$, $b \geq \frac{1}{2+\alpha} > \frac{1}{2+\beta}$, to derive a contradiction. ◀

C Other Candidate Algorithms

All the candidate algorithms are based on LPT. That is, whenever there is an idle machine, we always schedule the largest job. The only difference is the replacement rule. We will show that none of the them can beat the ratio of 1.5.

Candidate Algorithm 1. Fix any constant $0 < \rho < 1$. Upon the arrival of a job j, job k can be replaced by job j if k is the smallest processing job, $p_k < p_j$ and job k has been processed no larger than ρ fraction.

Counter example. At $t = 0$, m identical jobs come with $p_1 = p_2 = \cdots = p_m = 1$. Each of them is scheduled on a machine. At $t = \rho$, job $(m+1)$ comes with $p_{m+1} = 1 + \xi$ (ξ is an infinitesimal amount). Then one of jobs 1 to m is replaced by job $(m+1)$. At $t = 2\rho$, job $(m+2)$ comes with $p_{m+2} = 1 + 2\xi$, then job $(m+1)$ is replaced by job $(m+2)$. The same thing goes on and on, and at time $t = 1$, job $(m - 1 + \lceil \frac{1}{\rho} \rceil)$ is replaced by job $(m + \lceil \frac{1}{\rho} \rceil)$. After then, at $t = 1$, $(m - \lceil \frac{1}{\rho} \rceil)$ jobs come with $p_{m+\lceil \frac{1}{\rho} \rceil + 1} = p_{m+\lceil \frac{1}{\rho} \rceil + 2} = \cdots = p_{2m} = 1$. Then there are m pending jobs but only $(m-1)$ idle machines, so $\mathsf{ALG} = 3$. In the optimal solution, no replacement happens, and $\mathsf{OPT} = 2 + \lceil \frac{1}{\rho} \rceil \xi$.

Candidate Algorithm 2. Fix $0 < \rho < 1$ and $1 < \mu < 2$. Upon the arrival of a job j, job k can be replaced by job j if $\mu p_k < p_j$ and job k has been processed no larger than ρ fraction.

We show a counter example using $\mu = 3/2$ and $\rho = 1/2$. This counter example can be generalized to any μ and ρ such that $\mu + \rho \leq 2$.

Counter example. At $t = 0$, m jobs come, with $p_1 = m+1, p_2 = m+2, \cdots, p_m = 2m$. At time $t = m$, job $(m+1)$ comes with $p_{m+1} = 3m$. Then, as job m is the only job that is processed at most half, job m is replaced by job $(m+1)$. After the replacement, $(m-1)$ jobs come with $p_{m+2} = 2m+1, p_{m+3} = 2m+2, \cdots, p_{2m} = 3m-1$. Then $\mathsf{ALG} = 6m$, while in the optimal solution, no replacement happens, thus $\mathsf{OPT} = 4m+1$.

Candidate Algorithm 3. Fix a target performance ratio $1+\gamma$. When a job j comes, schedule it virtually to ALG, and calculate the current optimal solution with all the jobs that have been released. If the ratio can still be bounded in $1+\gamma$, do not replace any jobs; otherwise choose one job to replace.

Counter example. At $t = 0$, m jobs comes first, with $p_1 = 2m, p_2 = 2m+1, \cdots, p_m = 3m-1$. After each of them has been scheduled on a machine, at $t = 0$, another m jobs come, with $p_{m+1} = 3m, p_{m+2} = 3m+1, \cdots, p_{2m} = 4m-1$. At this time, since the local ALG and local OPT are exactly the same, no replacement happens. Then at $t = 3m-1$, another job comes with $p_{2m+1} = 3m$. So for this instance, $\mathsf{ALG} = 6m-1$; while in the optimal solution, three smallest jobs (jobs 1, 2, 3) are scheduled on the same machine, while all other jobs are paired up with the smallest with largest, and $\mathsf{OPT} = 4m+3$.

For our algorithm LPT with Restart, some may wonder what happens if α or β is not in $(0, 1/2)$. we know that if $\alpha = 0$, it is exactly the same with LPT, which cannot beat 1.5; if $\beta = 0$, a counter example can be given that is similar to that of Candidate Algorithm 1. Next we show that when $\beta \geq 1/2$ or $\alpha \geq 1/2$, LPT with Restart could not beat 1.5, no matter what the value of the other parameter is.

Candidate Algorithm 4. In LPT with Restart, set $\beta \geq 1/2$, and α to be any constant.

Counter example. At $t = 0$, $m (m \geq 4)$ jobs come first, with $p_1 = p_2 = \cdots = p_m = 1$. After all these jobs are scheduled, still at time 0, another m jobs come, with $p_{m+1} = p_{m+2} = \cdots = p_{2m} = 3/2 + \xi$. Then at $t = 1$, another job comes with $p_{2m+1} = 2$. Since $\beta \geq 1/2$, no replacement happens, and $\mathsf{ALG} = 9/2 + \xi$, while in optimal schedule, all jobs could be completed at or before time $3 + \xi$.

Candidate Algorithm 5. In LPT with Restart, set $\alpha \geq 1/2$, and β to be any constant such that $0 < \beta < 1/2$.

Counter example. We consider the special case with only one machine. At $t = 0$, job 1 comes with $p_1 = 1$. At $t = 1 - \xi$ (again, ξ is an infinitesimal amount), job 2 comes with $p_2 = 2$, then job 1 is replaced by job 2. At $t = 3 - 2\xi$, job 3 comes with $p_3 = 4$, job 2 is replaced by job 3. The same thing goes on and on. Each time a new job comes, ALG would replace the previous job, while OPT would wait for the previous job to end. The final ratio would be arbitrarily close to 1.5.

D Hardness for deterministic algorithms with restart

In this section, we present a simple lower bound of $\sqrt{1.5} \approx 1.225$ for any deterministic algorithms with restart. Given any deterministic algorithm, consider the following instance with two machines.

At $t = 0$, two jobs come with size $p_1 = p_2 = 1$. Then, at $t = 3 - \sqrt{6}$, another job comes with size $p_3 = \sqrt{6} - 1$.

1. If the algorithm starts processing job 3 after time 1, i.e., after completing the two jobs of size 1, no more jobs arrive in the instance. We have $\mathsf{OPT} = 2$ as we could have scheduled job 1 and job 2 on the same machine and job 3 on the other one. On the other hand, $\mathsf{ALG} \geq 1 + p_3 = \sqrt{6}$.
2. If the algorithm starts processing job 3 before time 1, e.g., it restarts one of the size-1 jobs, let there be a fourth job that arrives at time 1 with size $p_4 = \sqrt{6} - 1$. We have $\mathsf{OPT} = \sqrt{6}$ by scheduling job 1 and 3 on one machine, and 2 and 4 on the other. On the other hand, we have $\mathsf{ALG} \geq 3$ since at time 1 at least one of jobs 1 and 2 is pending, and job 3 does not complete until time 2.