

Flow Time Scheduling with Uncertain Processing Time

Yossi Azar*
azar@tau.ac.il
Tel Aviv University

Stefano Leonardi†
leonardi@diag.uniroma1.it
Sapienza University of Rome

Noam Touitou
noamtouitou@mail.tau.ac.il
Tel Aviv University

Abstract

We consider the problem of online scheduling on a single machine in order to minimize weighted flow time. The existing algorithms for this problem (STOC '01, SODA '03, FOCS '18) all require exact knowledge of the processing time of each job. This assumption is crucial, as even a slight perturbation of the processing time would lead to polynomial competitive ratio. However, this assumption very rarely holds in real-life scenarios.

In this paper, we present the first algorithm for weighted flow time which do *not* require exact knowledge of the processing times of jobs. Specifically, we introduce the Scheduling with Predicted Processing Time (SPPT) problem, where the algorithm is given a prediction for the processing time of each job, instead of its real processing time. For the case of a constant factor distortion between the predictions and the real processing time, our algorithms match *all* the best known competitiveness bounds for weighted flow time – namely $O(\log P)$, $O(\log D)$ and $O(\log W)$, where P , D , W are the maximum ratios of processing times, densities, and weights, respectively. For larger errors, the competitiveness of our algorithms degrades gracefully.

*Supported in part by the Israel Science Foundation (grant No. 2304/20 and grant No. 1506/16).

†Supported by the ERC Advanced Grant 788893 AMDROMA “Algorithmic and Mechanism Design Research in Online Markets” and MIUR PRIN project ALGADIMAR “Algorithms, Games, and Digital Markets”.

1 Introduction

The field of online scheduling focuses on efficient processing of jobs by machines, where the jobs are not known in advance but are released over time. An algorithm in this setting must assign each job to a machine, which must then process the job for some specific amount of time (called the *processing time* of the job).

In a classic setting for online scheduling, the algorithm is given a single machine, and aims to minimize the *total flow time*: the sum over jobs of the time that the job was pending in the algorithm (i.e. the difference between its completion time and its release time). In this setting, we usually allow *preemption* of a currently-processed job, which is pausing its processing until some future time, in which the processing will resume from the state in which it was paused. A classic result by Smith from 1956 [20] shows that this problem can be solved optimally using the SRPT (shortest remaining processing time) schedule.

A natural generalization of the total flow time metric is total *weighted* flow time, in which each job also has a *weight*. The goal is therefore to minimize the weighted sum of flow times, where the weight of a job is the weight of its flow time in the goal function. While natural, this problem has proved much harder than minimizing unweighted flow time: the best known algorithms [3, 1] have competitive ratios with logarithmic dependence on various parameters of the input; such parameters include the ratio P of the largest to smallest processing time in the input, the ratio W of the largest to smallest job weight in the input, and the ratio D of the largest to smallest density in the input (where the density of a job is the ratio of its weight to its processing time). As shown by Bansal and Chan [2], a dependence on these parameters is necessary (without added leniency such as speed augmentation).

However, these algorithms for both weighted and unweighted flow time make the assumption that the processing time of a job becomes known to the algorithm upon the release of the job. This assumption almost never holds in real-world scenarios, as nearly all computer programs of some complexity have varying running times. In the case of weighted flow-time, this assumption is ubiquitous – it is crucial to all known algorithms of sub-polynomial competitiveness [8, 3, 1], as replacing the real processing time with the predicted processing time in those algorithms would yield polynomial competitive ratios (e.g. $\Omega(P)$) even for a slight misprediction by a factor of $1 + \epsilon$ (compare this to the original polylogarithmic competitiveness of those algorithms).

Predicted Processing Time and Robustness. While knowing the exact processing time is infeasible in a real-world scenario, one could hold a *prediction* for the processing time of a job, of varying accuracy. Such predictions might be obtained from machine-learning algorithms, simple heuristics, or any other source. A good algorithm in this setting would be able to perform well given only the prediction, rather than the actual processing time, upon the release of a job.

The fact that the known guarantees for scheduling problems apply only for algorithms that know the processing time completely creates a gap between theory and practice. In this paper, we address this gap. Namely, we introduce the model of *scheduling with predicted processing time* (SPPT), in which upon the release of a job, we are provided with a prediction of the job’s processing time. An instance of the problem is characterized by a *distortion parameter*, which is the maximum ratio over jobs of the job’s real processing time and its predicted processing time.

In this paper, we present competitive algorithms which are μ -robust; that is, they maintain their competitive ratio guarantees for all inputs with distortion parameter at most μ . The competitive ratios have a polynomial dependency on μ (contrast with previously-known algorithms, where the competitive ratio immediately degrades to e.g. $\Omega(P)$ even when $\mu = 1 + \epsilon$ for a small ϵ). We assume that a value for the single parameter μ that bounds the distortion of the input can be learned, in order to apply the correct robust

algorithm; the case in which this assumption does not hold seems surprisingly tricky, and is discussed in Section 5.

At this point, one might ask whether the definition of the distortion parameter as the *maximum* distortion is indeed the correct measure for the error of the prediction; for example, could guarantees in some average of errors, e.g. geometric, be given? However, our choice of maximum distortion is not overly-conservative, but rather prescribed by the nature of the flow-time metric – specifically, due to its sensitivity to bad local competitive ratio. Consider for example, a scheduling instance which initially releases jobs with very erroneous, which would cause the algorithm to perform badly at some point in time. The adversary could then maintain this bad perform by releasing a stream of very short jobs; this well-known “bombardment” technique would guarantee a bad (global) competitive ratio for the algorithm. However, the processing time of these short “bombardment” jobs could be predicted with full accuracy, yielding a geometric average of errors which is arbitrarily close to 1 as the “bombardment” continues; this would also be the case for every other reasonable average.

1.1 Our Results

The SPPT Problem. We present algorithms for SPPT for minimizing weighted flow time. An algorithm, coupled with some competitiveness guarantee, is called μ -robust if its competitiveness guarantee is achieved for any instance with distortion parameter at most μ (and not only for inputs where the prediction matches the real processing time).

Without loss of generality, we assume one-sided error: that is, that the real processing time of each job is at least the predicted processing time, and at most μ times the predicted processing time. This one-sided error can be trivially obtained from a two-sided error with distortion parameter μ' by dividing each prediction by the distortion parameter μ' ; the new distortion parameter μ of the new one-sided-error instance would be at most $(\mu')^2$.

Recall that P, D, W are the ratios of maximum to minimum processing time, density and weight, respectively. For any μ , our results are:

1. A μ -robust algorithm for SPPT with weighted flow time, which is simultaneously $O(\mu^3 \log(\mu P))$ -competitive and $O(\mu^3 \log(\mu D))$ -competitive.
2. A μ -robust algorithm for SPPT with weighted flow time, which is $O(\mu^2 \log W)$ -competitive.

In the process of designing the $O(\mu^2 \log W)$ -competitive algorithm, we also design a μ -robust algorithm for SPPT with *unweighted* flow time, which is $2\lceil\mu^2\rceil$ -competitive. Note that, somewhat surprisingly, even for μ arbitrarily close to 1, there exists a lower bound of 2 on the competitiveness of any deterministic μ -robust algorithm, which we show in Appendix C.

For the weighted setting, and for a constant μ , our algorithms match the best known results in terms of all three parameters: the $O(\log W)$ -competitive algorithm of Bansal and Dhamdhere [3], and the $O(\log P)$ and $O(\log D)$ competitive algorithms in [1]. Even for distortion significantly larger than constant, e.g. $\mu = \Theta(\log P)$, our algorithms still obtain polylogarithmic competitiveness.

A special case of SPPT is the problem of semicclairvoyant scheduling [7, 6]. In this problem, instead of getting the processing time p_q of a job q , we are given its *class*, which is $\ell_q = \lceil \log_\rho p_q \rceil$ for some constant $\rho > 1$. Applying our algorithms for SPPT we obtain the first semicclairvoyant algorithms for weighted flow time, which match the best known guarantees for the *clairvoyant* setting. As a side result, we also obtain an improvement to the best known result for unweighted flow time. The exact statement of our results for the semicclairvoyant setting are given in Appendix D.

Paper Structure. The μ -robust algorithm with logarithmic dependency on either P or D is presented in Section 3. The μ -robust algorithm with logarithmic dependency on W is shown in Section 4. Section 4 comprises two subsections: Subsection 4.1 presents a μ -robust algorithm for the unweighted setting, while Subsection 4.2 uses the algorithm of Subsection 4.1 to construct the algorithm for the weighted setting. Appendix C shows a lower bound of 2-competitiveness for μ -robust algorithms in the unweighted setting, even for vanishingly small distortion. Appendix D discusses the application of the algorithms for SPPT to the semicclairvoyant setting.

1.2 Our Techniques

In developing robust algorithms for SPPT, we have used orthogonal methods and analyses for the algorithm for P and D and the algorithm for W . Curiously, it seems that the techniques of each single algorithm could not be applied to obtain the results of the other algorithm.

The $O(\mu^3 \log(\mu P))$ -competitive algorithm for SPPT, which is also $O(\mu^3 \log(\mu D))$ -competitive, is a simply-stated and novel algorithm. The flavor of its analysis is somewhat reminiscent of the $O(\log^2 P)$ -competitive algorithm of Chekuri *et al.* [8] (though, of course, our analysis shows an improved competitiveness, as our guarantee is $O(\log P)$ when μ is constant).

The algorithm begins by rounding the weights of jobs up to powers of $\Theta(\mu)$; this is the only place in the algorithm where μ is used. In weighted flow time, an algorithm must choose whether to prioritize high-weight jobs or cost-effective jobs (high density). The algorithm contends with this issue in the following way: it chooses the maximum weight w of a living job, and then chooses a job from the highest density class such that this class contains at least w weight. Inside this density class, the algorithm always chooses the highest weight job.

In the analysis of this algorithm, we show that the maximum weight in the algorithm never exceeds the *total* weight in the optimal solution. As for the lower weight classes which the algorithm *does* use, a volume-based analysis shows that the total weight in those classes is bounded.

The $O(\mu^2 \log W)$ -competitive algorithm for SPPT with weighted flow time is completely different in both methods and analysis from the previous algorithm, instead using the analysis framework introduced by Bansal and Dhamdhere [3].

The algorithm is constructed in two steps. First, we introduce a μ -robust algorithm for minimizing *unweighted* flow time. This algorithm uses two bins to maintain pending jobs, a *full* bin and a *partial* bin, which contain roughly the same number of jobs. Only jobs in the partial bin are processed by the algorithm, so that the jobs in the full bin retain their original processing time. Thus, the full bin provides a “counterweight” of full jobs to the partial bin; the intuition for this is that the algorithm wants to limit the fraction of partially-processed jobs at any point in time. Indeed, naive processing without attempting to limit the number of partially-processed jobs would result in unbounded competitive ratio, as observed in [6]. The competitiveness proof of this algorithm bears similarities to the proof framework of Bansal and Dhamdhere [3], also found in [1]. However, both of those papers use SRPT as a crucial component in their algorithms and analyses, which is infeasible in SPPT. Thus, our algorithms are designed to bypass this requirement.

When designing the priority of jobs in the full bin, we need to overcome the fact the algorithm does not know the remaining processing times of jobs (prohibiting strategies such as SRPT). Instead, the algorithm identifies and eliminates *violations*, which are pairs of jobs *provably* not ordered according to SRPT; this occurs when their predicted processing times differ by more than μ . The algorithm solves violations using a *rotation* operation, which identifies all violations involving a job and rotates their priorities. The resulting priority sequence, which contains no violations, is “*quasi-SRPT*” – that is, the upper bound for the

processing time of a low-priority job is always higher than the lower bound for the processing time of a high-priority job.

Interestingly, the priority among the jobs in the *partial* bin is LIFO, which is perhaps counterintuitive. While we are not sure that LIFO is the only option for achieving competitiveness, our proof crucially requires this feature.

We use the robust algorithm for unweighted flow time as a component in the $O(\mu^2 \log W)$ -competitive algorithm for SPPT with weighted flow time. This algorithm is composed of multiple full-partial bin pairs, one for each of the $\log W$ weight classes. Each bin pair behaves as an instance of the unweighted algorithm, and the algorithm chooses the heaviest bin pair to process at any given time (more specifically, the bin pair with the heaviest partial bin is processed). The proof framework used for the unweighted algorithm is now utilized, as it naturally supports multiple bin pairs.

One could hope that *any* μ -robust algorithm for the unweighted setting could be extended (at a loss of $O(\log W)$) to the weighted setting through binning. However, this is not known to be the case for weighted flow time – the binning technique is **not** black-box, and previous algorithms using binning ([3, 1]) demanded specific properties from the algorithm used in each bin. The main focus in designing the unweighted component of our algorithm is for it to “play nice” with the binning technique; this fact requires a delicate assignment of job priorities.

1.3 Related Work

The first algorithm with polylogarithmic guarantee for minimizing weighted flow time on a single machine was presented by Chekuri *et al.* [8], which was $O(\log^2 P)$ -competitive. Bansal and Dhamdhere [3] gave an $O(\log W)$ -competitive algorithm. Bansal and Chan [2] then showed that any deterministic algorithm for the problem must be $\Omega\left(\min\left\{\sqrt{\frac{\log W}{\log \log W}}, \sqrt{\frac{\log \log P}{\log \log \log P}}\right\}\right)$ -competitive, showing that dependence on at least one parameter is necessary. In [1], $O(\log P)$ -competitive and $O(\log D)$ -competitive algorithms were given, where D is the maximum ratio of densities of jobs (where density is processing time over weight). For more than a single machine, the problem is essentially intractable – Chekuri *et al.* [8] show a lower bound of $\Omega\left(\min\left\{\sqrt{P}, \sqrt{W}, (n/m)^{\frac{1}{4}}\right\}\right)$. With $(1 + \epsilon)$ speed augmentation, this problem is significantly easier [11, 3, 21].

The field of combining online algorithms with machine-learned predictions has seen significant interest in the last few years, with some works involving predictions in scheduling. For example, Purohit *et al.* [19] studied minimizing job completion time with predicted processing time. Other forms of prediction are also used; for example, Lattanzi *et al.* [12] studied the case of restricted assignment to multiple machines to minimize makespan, and used an algorithm-specific prediction (machine weights). Scheduling with predictions has also been studied in the queueing theory setting [16, 15]. Additional work on algorithms with predictions can be found in [13, 14, 17]

The case in which no prediction is given (and thus the algorithm knows nothing about the processing time of jobs) is called the *nonclairvoyant* model, and was studied in [18, 5, 4, 10, 9]. For minimizing unweighted flow time on a single machine, an $O(\log n)$ -competitive randomized algorithm is given in [5], where n is the number of jobs. This matches the lower bound of $\Omega(\log n)$ for randomized algorithms in [18]. Note that randomization is needed, and deterministic algorithms cannot get a sub-polynomial guarantee [18]. We are not aware of any nonclairvoyant results for minimizing weighted flow time.

Bechetti *et al.* [6] presented a 13-competitive algorithm for the *semclairvoyant* model, with the goal of minimizing unweighted flow time. The semclairvoyant setting was also considered by Bender *et al.* [7] for minimizing stretch (where the stretch of a job is the ratio of its flow time to its length)

2 Preliminaries

We first formalize the scheduling problem we consider in this paper. Then, we formalize the SPPT model, and the prediction given to the algorithm.

The Scheduling Problem. In the scheduling problem we consider, jobs arrive over time. Each job has its own processing time, which is the time it must be processed by the machine in order to be completed. The algorithm may choose at any point in time which job to process, and is allowed preemption.

An input consists of a set of jobs Q . Each job $q \in Q$ has the following properties:

1. A *processing time* $p_q > 0$.
2. A *weight* $w_q > 0$.
3. A *release time* r_q .

The goal of the algorithm ALG is to minimize the weighted flow time $F^{\text{ALG}} := \sum_{q \in Q} w_q \cdot (c_q^{\text{ALG}} - r_q)$ where c_q^{ALG} is the time in which request q is completed in the algorithm. An equivalent definition, which is the dominant one in this paper, is $F^{\text{ALG}} = \int_0^\infty \left(\sum_{q \in Q^{\text{ALG}}(t)} w_q \right) dt$ where $Q^{\text{ALG}}(t)$ is the set of pending jobs in the algorithm at time t .

The SPPT Model. In the SPPT model, the previous scheduling problem has the following modification. When a job q arrives, the algorithm does not become aware of p_q . Instead, it is given \tilde{p}_q , the predicted processing time of the job.

An algorithm for SPPT, coupled with a competitiveness guarantee, is called μ -robust if it maintains its competitiveness guarantee for inputs in which $\tilde{p}_q \leq p_q < \mu \tilde{p}_q$ for every job q in the input.

Notation. We denote by $y_q(t)$ the *remaining processing time* of q at t – that is, p_q minus the amount of time already spent processing q until time t . This amount is also called the *volume* of q at t .

For a set of jobs Q' , we define:

- $w(Q') := \sum_{q \in Q'} w_q$.
- $p(Q') := \sum_{q \in Q'} p_q$.
- $y(Q', t) := \sum_{q \in Q'} y_q(t)$.

For any time t :

- We define $Q(t)$ to be the set of pending jobs at t in the algorithm.
- We define $\delta(t) := |Q(t)|$ to be the number of pending jobs at t in the algorithm.
- We define $V(t) := y(Q(t), t)$ to be the total volume of pending jobs at t in the algorithm.
- We define $W(t) := w(Q(t))$ to be the total weight of pending jobs at t in the algorithm.

Fixing the optimal solution OPT for the given input, we refer to the attributes of OPT using the superscript $*$ (e.g. $y_q^*(t)$, $Q^*(t)$, $\delta^*(t)$, $V^*(t)$, $W^*(t)$).

3 Weighted Flow Time – Ratio of Processing Times

In this section, we present a μ -robust algorithm which is $O(\mu^3 \log(\mu P))$ -competitive.

The main result of this section is the following theorem.

Theorem 3.1. For every μ , there exists a μ -robust algorithm for the SPPT problem that is $O(\mu^3 \log(\mu P))$ -competitive.

3.1 The Algorithm

We start by rounding the weights of jobs up to powers of $\lambda = \Theta(\mu)$; specifically, we choose $\lambda = 16\mu + 6$. From now on, when referring to the weights of jobs, we refer to those rounded weights.

Definition 3.2 (weights and ID). Throughout this section we use the following definitions:

- We define $wc(q) := \log_\lambda(w_q)$ to be the *weight class* of a request q (which is an integer, due to the rounding of weights).
- We define $u_i := \lambda^i$ for every integer i .
- We define $idc(q) := \left\lfloor \log_2 \left(\frac{\bar{p}_q}{w_q} \right) \right\rfloor$ to be the *estimated inverse density class* of a job q , abbreviated as EI-density class (the amount $\frac{\bar{p}_q}{w_q}$ is called the EI-density of q).

Definition 3.3 (predicated jobs). For every two predicates p_1, p_2 we define

$$Q_{p_1, p_2}(t) = \{q \in Q(t) \mid p_1(wc(q)) \text{ and } p_2(idc(q))\}$$

Similarly, we define $W_{p_1, p_2}(t) = w(Q_{p_1, p_2}(t))$ and $V_{p_1, p_2}(t) = y(Q_{p_1, p_2}(t), t)$.

For example, $Q_{\leq i, =j}(t)$ is the set of all living jobs q at t such that $wc(q) \leq i$ and $idc(q) = j$. We use \top to denote the predicate that always evaluates to true; it is used when we would like to take jobs of every EI-density/weight.

We say that a job is *partial* in the algorithm if it has been processed for any amount of time; otherwise, the job is *full*.

Description of the Algorithm. The algorithm we describe prefers high weight jobs and high density jobs, and attempts to balance these two considerations. This is done by processing some job of the maximum possible density, i.e. minimum EI-density, subject to the total weight of jobs from that EI-density class exceeding the weight of the heaviest living job. After choosing the correct EI-density class in this manner, the algorithm must choose a job from that EI-density class; it chooses from the heaviest weight class in which there is a living job. Once the correct EI-density and weight classes have been chosen, the algorithm prefers partial jobs, thus maintaining the fact that there is at most one partial job of each EI-density-weight combination.

The formal description of the algorithm appears in Algorithm 1.

Algorithm 1: Scheduling with Predictions – Weighted

- 1 **Event Function** PROCESS() // at any point in time t
- 2 Let i be the largest weight class in which a job is alive.
- 3 Let j be the minimal EI-density class such that $W_{\top, =j}(t) \geq u_i$.
- 4 Let i' be the maximum weight class in which a job is alive in $Q_{\top, =j}(t)$.
- 5 Process a job from $Q_{=i', =j}(t)$ (preferring a partial job if exists).

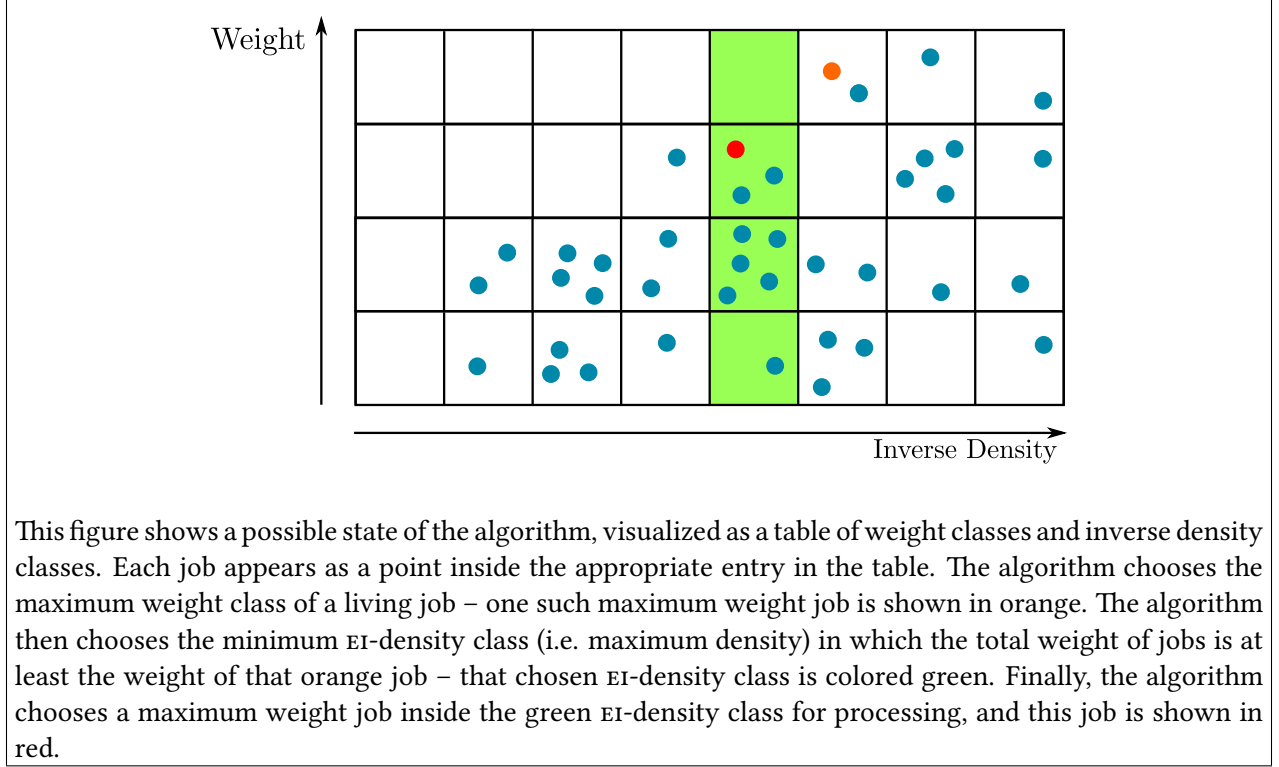


Figure 1: The State and Operation of Algorithm 1

3.2 Analysis

Consider an instance in which the distortion parameter is at most μ . Observe that the ratio between the maximum predicted processing time and the minimum predicted processing time is at most μP .

Fixing any weight u_i , observe that a job of weight u_i can only belong to EI-density classes in a limited range, which contains at most $\lceil \log(\mu P) \rceil + 1$ classes. We henceforth define $\Lambda := \lceil \log(\mu P) \rceil + 1$, the maximum number of EI-density classes to which jobs of a specific weight can belong.

The main lemma used to prove Theorem 3.1 is the following, which states that the algorithm is locally competitive.

Lemma 3.4. *At any point in time t , it holds that $W(t) \leq O(\Lambda \mu^2) \cdot W^*(t)$.*

The main focus henceforth would be on proving Lemma 3.4.

Definition 3.5 (analogue of Definition 3.3 for OPT). For every two predicates p_1, p_2 we define

$$Q_{p_1, p_2}^*(t) = \{q \in Q^*(t) \mid p_1(\text{wc}(q)) \text{ and } p_2(\text{idc}(q))\}$$

We define $W_{p_1, p_2}^*(t) = w(Q_{p_1, p_2}^*(t))$ and $V_{p_1, p_2}^*(t) = y^*(Q_{p_1, p_2}^*(t), t)$.

In addition, we use Δ to denote to volume difference between the algorithm and the optimal solution; for example, $\Delta V_{\leq i, =j}(t) = V_{\leq i, =j}(t) - V_{\leq i, =j}^*(t)$.

Definition 3.6 (important weight classes). We define the following:

1. We define $\text{cwc}(t)$ to be the weight class of the job processed by the algorithm at time t .
2. We define $\text{clw}(t)$ to be the largest weight class of a living job in the algorithm at t .

3. We define $i^*(t)$ to be the minimum weight class such that $u_{i^*(t)} > \frac{4\mu}{\lambda} \cdot W^*(t)$.

The following observation is immediate from the above definition.

Observation 3.7. *There is no job of weight class more than $i^*(t)$ alive in OPT at time t .*

Henceforth, fix a point in time t for proving Lemma 3.4. For brevity, we also write i^* instead of $i^*(t)$.

The following lemma allows converting weight to volume, and its proof appears in Appendix A.

Lemma 3.8. *Let i be a weight class, and let j_1, j_2 be two EI-density classes such that $j_1 \leq j_2$. Then*

$$\sum_{j=j_1}^{j_2} W_{\leq i, =j}(t) \leq \min\{2(j_2 - j_1 + 1), 4\Lambda\} \cdot u_i + 2\mu W^*(t) + 2 \sum_{j=j_1}^{j_2} \max\left\{0, \frac{\Delta V_{\leq i, \leq j}(t)}{2^j}\right\}$$

Definition 3.9. Define t_j to be the last time prior to t in which the algorithm processed a job of weight class at most i^* and EI-density class strictly more than j .

Proposition 3.10 shows that the algorithm has low maximum weight at t_j for every j , and bounds differences in volume between the algorithm and the optimal solution (which enables use of Lemma 3.8). The proof of Proposition 3.10 is given in Appendix A.

Proposition 3.10. *Let j be an EI-density class, then it holds that:*

1. $\text{clw}(t_j) \leq i^*$
2. $\Delta V_{\leq i^*, \leq j}(t) \leq \mu \cdot 2^{j+2} u_{\text{clw}(t_j)}$

Proof of Lemma 3.4. First, we claim that $\text{clw}(t) \leq i^*$. Assume otherwise, and observe the last time t' in which the algorithm processed a job of weight at most i^* . It must be that $\text{clw}(t') > i^*$; this is since the optimal solution has no jobs of weight class more than i^* alive at t , and since the algorithm works only on such jobs from t' onwards. However, t' is also equal to t_j for some j , which contradicts Proposition 3.10. Thus, it must be that $\text{clw}(t) \leq i^*$.

Let j_{\min} be the minimum EI-density class in which a job of weight class at most i^* can exist, and let j_{\max} be the maximum possible EI-density class. It holds that

$$\begin{aligned} W(t) &= \sum_{j=j_{\min}}^{j_{\max}} W_{\leq i^*, =j}(t) \\ &\leq 4\Lambda u_{i^*} + 2\mu W^*(t) + \sum_{j=j_{\min}}^{j_{\max}} 2 \max\left\{0, \frac{\Delta V_{\leq i^*, \leq j}(t)}{2^j}\right\} \\ &\leq 4\Lambda u_{i^*} + 2\mu W^*(t) + \sum_{j=j_{\min}}^{j_{\max}} 8\mu \cdot u_{\text{clw}(t_j)} \end{aligned}$$

where the equality is due to $\text{clw}(t) \leq i^*$, the first inequality is due to Lemma 3.8 and the second inequality follows from Proposition 3.10. Now, consider that for every EI-density class j , at t_j we processed a job q of EI-density class h where $h > j$, such that $\text{wc}(q) \leq \text{clw}(t_j)$. There also existed at t_j a job q' such that $\text{wc}(q') = \text{clw}(t_j)$; denote its EI-density class $h' := \text{idc}(q')$. It holds that $h' \geq h$, otherwise q' would have been chosen for processing at t_j instead of q . Thus, it must be the case that an EI-density class larger than j is one of the Λ EI-density classes to which $\text{clw}(t_j)$ can belong.

From the preceding argument, $i^* = \text{clw}(t_j)$ can hold for at most Λ values of j ; $i^* - 1 = \text{clw}(t_j)$ can

hold for at most Λ additional values of j ; and so on. It thus holds that:

$$8\mu \sum_{j=j_{\min}}^{j_{\max}} u_{\text{clw}(t_j)} \leq 8\mu\Lambda \sum_{k=0}^{\infty} u_{i^*-k} \leq 16\mu\Lambda u_{i^*}$$

Therefore, we have

$$W(t) \leq \Lambda(16\mu + 4)u_{i^*} + 2\mu W^*(t) \quad (1)$$

Observe that $u_{i^*} \leq 4\mu W^*(t)$ – this stems from the definition of i^* . Plugging into Equation (1), we get that

$$W(t) \leq O(\Lambda\mu^2)W^*(t) \quad \square$$

Proof of Theorem 3.1. Plugging the definition of Λ into Lemma 3.4, we have that

$$W(t) \leq O(\mu^2 \log(\mu P))W^*(t)$$

Recall that this is after the rounding of the weights to powers of $\lambda = \Theta(\mu)$; taking this rounding into account, the algorithm is $O(\mu^3 \log(\mu P))$ -competitive. \square

3.3 Ratio of Densities

Recall that the density of a job q is $\frac{w_q}{p_q}$. We show that the dependence on P in the competitiveness of Algorithm 1 (as stated in Theorem 3.1) can be replaced with a dependence on the parameter D , which is the maximum ratio of the densities of two jobs in the input. Concretely, we show that the following theorem holds.

Theorem 3.11. *Algorithm 1 is a μ -robust, $O(\mu^3 \log(\mu D))$ -competitive algorithm for the SPPT problem.*

The proof of Theorem 3.11 appears in Appendix E, and is almost immediate from the previous proof of Theorem 3.1.

4 Weighted Flow Time – Ratio of Weights

In this section, we present the $O(\mu^2 \log W)$ -competitive μ -robust algorithm for SPPT with weighted flow time. We do so in two steps: first, we introduce μ -robust algorithm for the unweighted setting, which is $2\lceil\mu^2\rceil$ -competitive. Then, we show that instances of this unweighted algorithm can be combined and applied to the weighted setting, at an additional loss of an $O(\log W)$ factor in competitiveness.

4.1 The Algorithm for Unweighted Flow Time in the Prediction Model

In this subsection, we present an algorithm in the prediction model for the unweighted setting – that is, where $w_q = 1$ for all $q \in Q$.

4.1.1 The Algorithm

The algorithm holds each pending job in one of two bins, a “full” bin F and a “partial” bin P . The algorithm attempts to divide the total weight of the pending jobs equally between the two bins. The algorithm refers to the pending jobs in F by Q_F and the pending jobs in P by Q_P . For ease of notation, we also define $\delta_F = |Q_F|$ and $\delta_P = |Q_P|$.

The course a job undergoes in the algorithm consists of being initially assigned to bin F , then moving to bin P , and finally being processed in bin P until its eventual completion. The algorithm maintains a bijective priority mapping $\pi_F : Q_F \rightarrow \{1, 2, \dots, \delta_F\}$ from the living jobs Q_F to the natural integers from 1 to δ_F . This mapping determines the priority of the jobs of F . That is, whenever moving a job from F to P is required, the algorithm moves the highest priority job (q such that $\pi_F(q) = \delta_F$).

The algorithm also maintains (implicitly) a priority mapping $\pi_P : Q_P \rightarrow \{1, 2, \dots, \delta_P\}$, such that the highest priority job is chosen to be processed. This priority mapping is simply LIFO according to the time in which the job moved from F to P .

Definition 4.1 (violation). Observe two jobs $q_1, q_2 \in Q_F$ in the algorithm. We say that the ordered pair (q_1, q_2) is a *violation* if $\pi_F(q_1) > \pi_F(q_2)$ and $\mu \cdot \tilde{p}_{q_2} \leq \tilde{p}_{q_1}$. Informally, the algorithm assigns priority to q_1 over q_2 even though it knows that the processing time of q_2 (which is strictly smaller than $\mu \cdot \tilde{p}_{q_2}$) is strictly smaller than that of q_1 .

Algorithm’s Description and Intuition. The algorithm maintains two properties regarding the priorities in the bin F . The first property is that there exist no violations. When this holds, the priority is “**quasi-SRPT**”; that is, if a job has a higher priority than another job, then the lower bound on the processing time of the first job is no more than the upper bound on the processing time of the second job (noting that in F the remaining processing time is equal to the initial processing time). This “quasi-SRPT” property is used in Proposition B.8. The second property is that for every natural k , the total processing time of the k lowest priority jobs never decreases upon the release of a new job, which is used in Proposition B.6.

The way in which the algorithm maintains both of those traits for the bin F can be seen in the function `UPONJOBRELEASE(q)`, which is called upon the release of a new job q . The algorithm first gives the newly-released job the maximum priority in F , which does not affect the second property, but could possibly break the first property by causing violations. In order to fix these violations, the algorithm performs a **rotation** of the new job q and all violations. That is, numbering the jobs in violation with q as q_1, \dots, q_m by order of decreasing priority, the rotation simultaneously sets the new priority of q to be the old priority of q_m , the new priority of q_m to be the old priority of q_{m-1} , and so on (the new priority of q_1 is the old priority of q , i.e. the maximum priority). In Algorithm 2, this rotation operation is described as simply composing the mapping π_F with the cyclic permutation $\sigma = (\pi_F(q_m), \pi_F(q_{m-1}), \dots, \pi_F(q_1), \pi_F(q))$, where σ maps from the natural numbers $[\delta_F]$ to $[\delta_F]$, and is written in cycle notation. In the analysis, we show that this rotation is sufficient to solve any violations that occur in the insertion.

Visualization in Figures. Visualizations of the state of Algorithm 2 are given throughout the paper (e.g. in Figure 2). In these visualizations, each job is visualized as a rectangle, where the height of the rectangle is the weight of the job (which is currently 1, since we’re considering the unweighted setting). The area of the rectangle is the remaining volume of the job. Consider the transformation of the rectangle as a job is processed: its weight (height) remains the same but its volume (area) decreases. Hence, the width of the rectangle decreases as the job is processed.

A bin (either F or P) is visualized as a “stack” of the jobs in that bin. Those jobs are stacked according to their priority in the bin (the highest priority job is at the top of the stack). At any point in time, the highest priority job in P is processed – this is the job at the top of the visualization of P .

These visualizations are for the sake of understanding the algorithm and its proof, and are not used in the algorithm itself. The algorithm is not aware of the details of the visualization, and in particular is not aware of the volume of the jobs (i.e. the area of the rectangles).

The algorithm is shown in Algorithm 2. Figures 2 and 3 visualize Algorithm 2.

The competitiveness guarantee of Algorithm 2 is given in Theorem 4.2, the proof of which is given in

Algorithm 2: Scheduling with Predictions – Unweighted

```

1 Event Function UPONJOBRELEASE( $q$ ) // upon the release of a job  $q$  at time  $t$ 
2   Set  $Q_F \leftarrow Q_F \cup \{q\}$ , and set  $\pi_F(q) \leftarrow \delta_F$ .
   // rotate the priorities of jobs to fix violations
3   if there exists  $q' \in Q_F \setminus \{q\}$  such that  $(q, q')$  is a violation then
4     let  $Q' = \{q' \in Q_F \setminus \{q\} \mid (q, q') \text{ is a violation}\}$ .
5     denoting  $m = |Q'|$ , let  $q_1, \dots, q_m$  be an enumeration of  $Q'$  such that
        $\pi_F(q) > \pi_F(q_1) > \dots > \pi_F(q_m)$ .
6     let  $\sigma$  be the cyclic permutation  $(\pi_F(q_m), \pi_F(q_{m-1}), \dots, \pi_F(q_1), \pi_F(q))$ .
7     set  $\pi_F \leftarrow \sigma \circ \pi_F$ 

8 Event Function UPONHEAVYF() // upon  $\delta_F > \delta_P$ 
9   let  $q \in Q_F$  be such that  $\pi_F(q) = \delta_F$ .
10  move  $q$  from  $F$  to  $P$ , and set  $\pi_P(q) = \delta_P$ .

11 Event Function PROCESS() // at any point in time  $t$ 
12  let  $q \in Q_P$  be such that  $\pi_P(q) = \delta_P$ .
13  process  $q$ .

```

Appendix B.1.

Theorem 4.2. *Algorithm 2 is a μ -robust, $2^{\lceil \mu^2 \rceil}$ -competitive algorithm for the unweighted SPPT problem.*

4.2 The $O(\mu^2 \log W)$ -competitive Algorithm for Weighted Flow Time

Having described the unweighted algorithm in the previous subsections, we present the $O(\mu^2 \log W)$ -competitive algorithm for the weighted setting.

4.3 The Algorithm

We assume that the weights of incoming jobs are of an integer power of 2; the algorithm can enforce this by rounding weights, incurring a factor of at most 2 to its competitive ratio.

For each $i \in \mathbb{Z}$, the algorithm maintains a *superbin* A^i (only nonempty superbins are maintained explicitly). The superbin is constructed as in Subsection 4.1. That is, A^i contains two bins F^i and P^i (containing the pending jobs Q_{F^i} and Q_{P^i} , respectively), and maintains the priority bijections π_{F^i} and π_{P^i} on the jobs of F^i and P^i , respectively. As in the weighted case, the algorithm refers to $|Q_{F^i}|$ and $|Q_{P^i}|$ by δ_{F^i} and δ_{P^i} , respectively.

The algorithm sends released jobs of weight 2^i (for any $i \in \mathbb{Z}$) to the superbin A^i . The insertion to the superbin A^i is according to UPONJOBRELEASE in Algorithm 2. That is, the released job q is first inserted to F^i , the algorithm sets $\pi_{F^i}(q) \leftarrow \delta_{F^i}$, and then possibly performs a rotation to solve violations. For every superbin index i , the algorithm also maintains that $w(Q_{F^i}) \leq w(Q_{P^i})$, by moving jobs from F^i to P^i as in Subsection 4.1, performed by calling UPONHEAVYF as defined in Algorithm 2.

As for processing, the algorithm chooses for processing the superbin A^i with the heaviest partial bin P^i . When a superbin A^i is chosen for processing, algorithm makes a call to PROCESS as defined in Algorithm 2, which chooses the job $q \in Q_{P^i}$ such that $\pi_{P^i}(q)$ is maximal.

The algorithm is given in Algorithm 3. A possible state of Algorithm 3 is visualized in Figure 4.

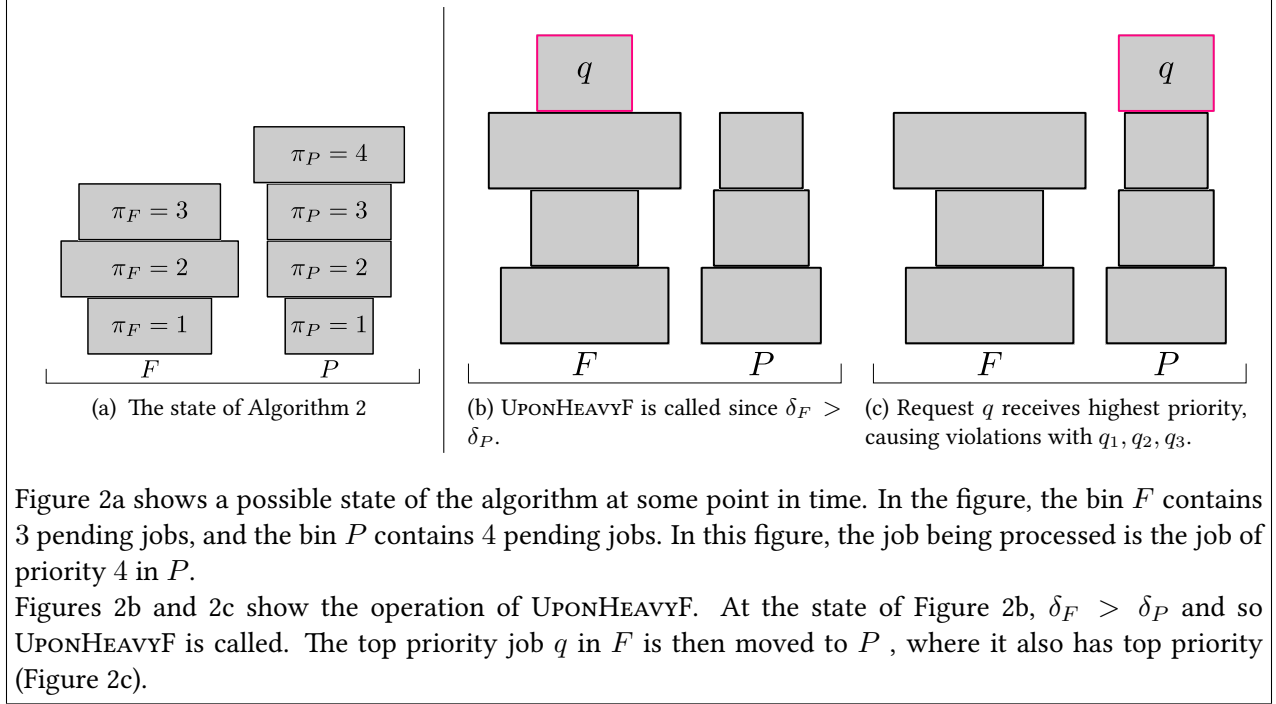


Figure 2: The State of Bins and UPONHEAVYF in Algorithm 2

The competitiveness guarantee of Algorithm 3 is given in Theorem 4.3, the proof of which appears in Appendix B.2.

Theorem 4.3. *Algorithm 3 is a μ -robust, $O(\mu^2 \log W)$ -competitive algorithm for the SPPT problem.*

5 Discussion and Open Problems

This paper presents algorithms for SPPT for minimizing weighted flow time which have a polynomial dependency on the distortion μ of the given predictions, such that when this distortion is constant the best known competitiveness bounds are matched (namely, $O(\log P)$, $O(\log D)$ and $O(\log W)$). These are the first algorithms to maintain *any* nontrivial competitiveness when given inaccurate processing times in the weighted flow time setting.

However, in the model which we consider, each such μ -robust algorithm is tailored to a specific value of μ . One could desire an algorithm which works for *any* μ , while maintaining a competitive ratio which is a function of this μ . In many other problems, going from the first model to the second model can be very easily done through a doubling procedure on the parameter μ ; however, applying such a doubling scheme to this scheduling problem does not seem immediate, and we leave its development to future work.

The dependencies of our algorithms on the maximum distortion are $O(\mu^2)$ and $\tilde{O}(\mu^3)$. Improving the dependence on μ in the competitiveness is also an interesting problem; we conjecture that a linear dependence on μ is the optimal one.

Another problem is determining the exact competitive ratio for μ -robust algorithm as μ approaches 1 (i.e. small distortion). This problem is most salient for unweighted flow time, in which we've shown a lower bound of 2-competitiveness, and our upper bound result only yields 4-competitiveness.

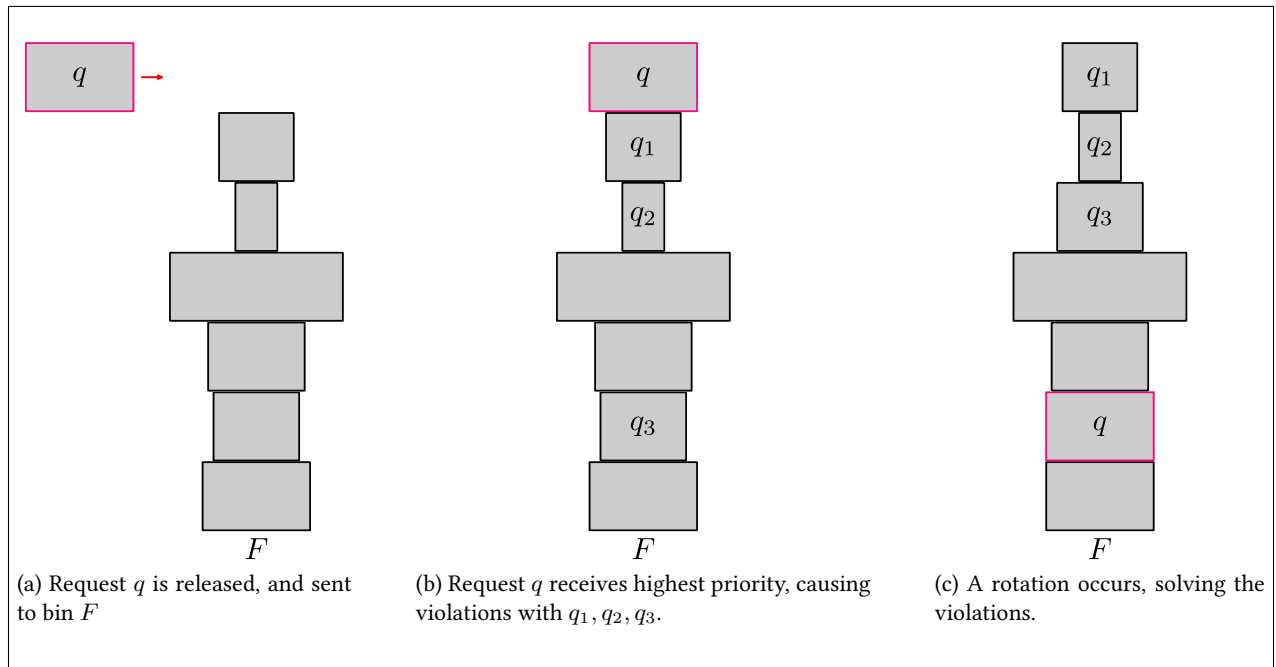


Figure 3: UPONJOBRELEASE in Algorithm 2

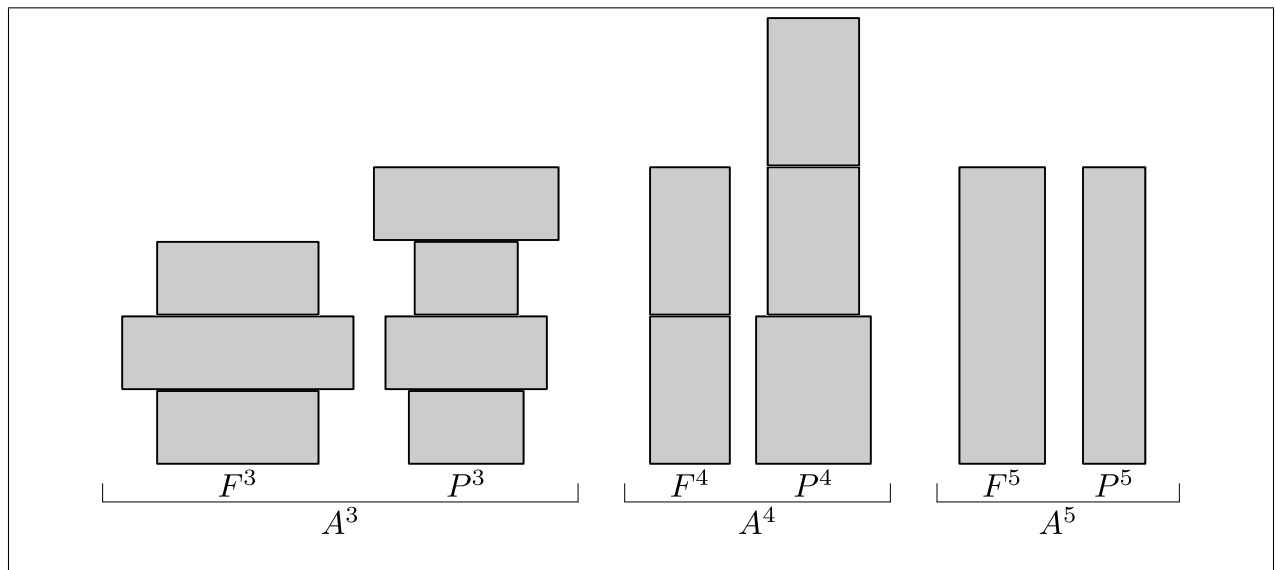


Figure 4: A Possible State of Algorithm 3

Algorithm 3: Scheduling with Predictions – $O(\log W)$ -Competitive

```
1 Event Function UPONJOBRELEASE( $q$ ) // upon the release of a job  $q$  at time  $t$ 
2   | let  $i$  be such that  $w_q = 2^i$ .
3   | call  $A^i$ .UPONJOBRELEASE( $q$ )
4 Event Function PROCESS() // at any point in time  $t$ 
5   | let  $i = \arg \max_{i'} w(P^{i'})$ .
6   | call  $A^i$ .PROCESS().
7 Event Function UPONHEAVYF( $i$ ) // upon  $w(F^i) > w(P^i)$  for some index  $i$ 
8   | call  $A^i$ .UPONHEAVYF()
```

References

- [1] Yossi Azar and Noam Touitou. Improved online algorithm for weighted flow time. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 427–437. IEEE Computer Society, 2018.
- [2] Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit $o(1)$ -competitive algorithms. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 1238–1244, 2009.
- [3] Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. *ACM Trans. Algorithms*, 3(4):39, 2007. also in SODA 2003: 508-516.
- [4] Nikhil Bansal, Kedar Dhamdhere, Jochen Könemann, and Amitabh Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. *Algorithmica*, 40(4):305–318, 2004.
- [5] Luca Becchetti and Stefano Leonardi. Non-clairvoyant scheduling to minimize the average flow time on single and parallel machines. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 94–103, 2001.
- [6] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Semi-clairvoyant scheduling. *Theor. Comput. Sci.*, 324(2-3):325–335, 2004.
- [7] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Improved algorithms for stretch scheduling. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 762–771. ACM/SIAM, 2002.
- [8] Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 84–93, 2001.
- [9] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. *Journal of the ACM (JACM)*, 65(1):1–33, 2017.
- [10] Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 531–540. IEEE, 2014.

- [11] Jae-Hoon Kim and Kyung-Yong Chwa. Non-clairvoyant scheduling for weighted flow time. *Inf. Process. Lett.*, 87(1):31–37, 2003.
- [12] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, New Orleans, LA, USA, January 5 - 8, 2020.*, 2020.
- [13] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3302–3311, 2018.
- [14] Andres Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1858–1866, 2017.
- [15] Michael Mitzenmacher. Queues with small advice. *CoRR*, abs/2006.15463, 2020.
- [16] Michael Mitzenmacher. Scheduling with Predictions and the Price of Misprediction. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [17] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *arXiv preprint arXiv:2006.09123*, 2020.
- [18] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47, 1994.
- [19] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.
- [20] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.
- [21] Jianqiao Zhu, Ho-Leung Chan, and Tak Wah Lam. Non-clairvoyant weighted flow time scheduling on different multi-processor models. *Theory Comput. Syst.*, 56(1):82–95, 2015.

A Additional proofs from Section 3

Proof of Lemma 3.8. Observe that $\sum_{j=j_1}^{j_2} W_{\leq i,=j}(t) = \sum_{j=j_1}^{j_2} W_{\leq i,=j}^p(t) + \sum_{j=j_1}^{j_2} W_{\leq i,=j}^f(t)$, where the superscripts p and f refer to partial and full jobs, respectively.

The algorithm maintains that in every weight class + EI-density class combination, there exists at most one partial job of those classes. Thus, for every EI-density class j , a geometric sum yields that $W_{\leq i,=j}^p(t) \leq 2w^j$, where w^j is the maximum weight of a partial job in $Q_{\leq i,=j}(t)$. Since $w^j \leq u_i$, we thus have that $W_{\leq i,=j}^p(t) \leq 2u_i$, and thus $\sum_{j=j_1}^{j_2} W_{\leq i,=j}^p(t) \leq 2(j_2 - j_1 + 1)u_i$. In addition, each weight class can exist in at most Λ different EI-density classes, which yields

$$\sum_{j=j_1}^{j_2} W_{\leq i,=j}^p(t) \leq \sum_{j=j_1}^{j_2} 2w^j \leq 2u_i\Lambda + 2u_{i-1}\Lambda + \dots \leq 4u_i\Lambda$$

Combining, we have $\sum_{j=j_1}^{j_2} W_{\leq i,=j}^p(t) \leq \min\{2(j_2 - j_1 + 1), 4\Lambda\} \cdot u_i$.

It remains to bound the weight of the full jobs, namely $\sum_{j=j_1}^{j_2} W_{\leq i,=j}^f(t)$.

$$\begin{aligned} \sum_{j=j_1}^{j_2} W_{\leq i,=j}^f(t) &\leq \sum_{j=j_1}^{j_2} \frac{V_{\leq i,=j}(t)}{2^j} \\ &\leq \sum_{j=j_1}^{j_2} \frac{\Delta V_{\leq i,=j}(t) + V_{\leq i,=j}^*(t)}{2^j} \\ &\leq 2\mu \sum_{j=j_1}^{j_2} W_{\leq i,=j}^*(t) + \sum_{j=j_1}^{j_2} \frac{\Delta V_{\leq i,\leq j}(t) - \Delta V_{\leq i,\leq j-1}(t)}{2^j} \\ &\leq 2\mu \sum_{j=j_1}^{j_2} W_{\leq i,=j}^*(t) + \frac{\Delta V_{\leq i,\leq j_2}(t)}{2^{j_2}} + \sum_{j=j_1}^{j_2-1} \frac{\Delta V_{\leq i,\leq j}(t)}{2^{j+1}} - \frac{\Delta V_{\leq i,\leq j_1-1}(t)}{2^{j_1}} \\ &\leq 2\mu \sum_{j=j_1}^{j_2} W_{\leq i,=j}^*(t) + \frac{\Delta V_{\leq i,\leq j_2}(t)}{2^{j_2}} + \sum_{j=j_1}^{j_2-1} \frac{\Delta V_{\leq i,\leq j}(t)}{2^{j+1}} + \mu W_{\leq i,\leq j_1-1}^*(t) \\ &\leq 2\mu \sum_{j=j_1}^{j_2} W_{\leq i,=j}^*(t) + \sum_{j=j_1}^{j_2} \max\left\{0, \frac{\Delta V_{\leq i,\leq j}(t)}{2^j}\right\} + \mu W_{\leq i,\leq j_1-1}^*(t) \\ &\leq 2\mu W_{\leq i,\leq j_2}^*(t) + \sum_{j=j_1}^{j_2} \max\left\{0, \frac{\Delta V_{\leq i,\leq j}(t)}{2^j}\right\} \\ &\leq 2\mu W^*(t) + \sum_{j=j_1}^{j_2} \max\left\{0, \frac{\Delta V_{\leq i,\leq j}(t)}{2^j}\right\} \end{aligned}$$

where the first inequality is due to the fact that a full job q of EI-density class j has at least $w_q \cdot 2^j$ remaining volume, the second and fifth inequalities are due to the fact that every job q of EI-density class at most j in the optimal solution has at most $2^{j+1}\mu w_q$ volume. \square

Proof of Proposition 3.10. First, we show that for every j , the first claim implies the second claim. Then, we prove the first claim by induction on the EI-density classes, in decreasing order. Combining these two proofs, the proposition holds.

The first claim implies the second claim. Assume that $\text{clw}(t_j) \leq i^*$ for some j . Since the optimal solution has no jobs of weight more than u_{i^*} alive at time t , and since the algorithm did have any such jobs alive at t_j , it must be that the algorithm worked on such jobs during $[t_j, t)$ at least as much as the optimal solution. In addition, the algorithm did not work on any jobs of weight at most u_{i^*} and EI-density class more than j during this interval. Thus, it holds that

$$\Delta V_{\leq i^*, \leq j}(t) \leq \Delta V_{\leq i^*, \leq j}(t_j) \leq V_{\leq i^*, \leq j}(t_j)$$

Thus, it's enough to bound $V_{\leq i^*, \leq j}(t_j)$. Observe that since the algorithm chose to process a job of EI-density class more than j , the total weight in each EI-density class $j' \leq j$ is at most $\text{clw}(t_j)$. Thus, the total volume at t_j in EI-density class j' is at most $\mu \cdot 2^{j'+1} u_{\text{clw}(t_j)}$. Summing over all $j' \leq j$ yields a geometric sum, which is at most $2^{j+2} u_{\text{clw}(t_j)}$, showing that the first claim indeed implies the second claim.

The first claim holds. We prove the first claim of the proposition by induction on the EI-density classes, in decreasing order. For the base case of the largest EI-density class j_{\max} , we have $t_{j_{\max}} = 0$ and thus the claim holds.

Now, let j be any EI-density class and suppose that the claim holds for all EI-density classes $j+1$ and above. Assume for contradiction that $\text{clw}(t_j) > i^*$. Let q be the job chosen for processing at t_j , and define $h := \text{idc}(q)$. From the definition of t_j , we know that $\text{wc}(q) \leq i^*$ and that $h > j$.

Since the algorithm always processes the maximum-weight job in the chosen EI-density class, we know that

$$W_{\leq \text{wc}(q), =h}(t_j) = W_{\top, =h}(t_j) \geq u_{\text{clw}(t_j)}$$

where the inequality is due to the choice of the algorithm.

Now, since during $[t_j, t)$ the algorithm did not process any jobs of weight class at most i^* and EI-density class more than j , it must be that

$$W_{\leq i^*, =h}(t) \geq W_{\leq i^*, =h}(t_j) = W_{\leq \text{wc}(q), =h}(t) \geq u_{\text{clw}(t_j)} - u_{i^*}$$

where the minus term is due to the fact that the algorithm could have completed q exactly at t_j .

Now, using Lemma 3.8, we have

$$\begin{aligned} W_{\leq i^*, =h}(t) &\leq 2 \cdot u_{i^*} + 2\mu W^*(t) + 2 \max \left\{ 0, \frac{\Delta V_{\leq i^*, \leq h}(t)}{2^h} \right\} \\ &\leq 2 \cdot u_{i^*} + 2\mu W^*(t) + 8\mu u_{\text{clw}(t_h)} \\ &\leq 2 \cdot u_{i^*} + 2\mu W^*(t) + 8\mu u_{i^*} \\ &= (8\mu + 2)u_{i^*} + 2\mu W^*(t) \end{aligned}$$

where the second inequality uses the induction hypothesis that the first claim holds for h , combined with the previous proof that the first claim implies the second claim. The third inequality also uses the induction hypothesis.

Combining, we have $2\mu W^*(t) \geq u_{\text{clw}(t_j)} - (8\mu + 3)u_{i^*} \geq u_{i^*+1} - (8\mu + 3)u_{i^*} = \frac{\lambda}{2} \cdot u_{i^*}$. Thus, we get that $W^*(t) \geq \frac{\lambda}{4\mu} \cdot u_{i^*}$, in contradiction to the definition of i^* . This proves the first claim of the proposition. \square

B Additional proofs from Section 4

B.1 Proof of Theorem 4.2

In the following analysis, we prove Theorem 4.2.

Definition B.1. For every time t , we define $Q_F(t)$ and $Q_P(t)$ to be the values of the variables Q_F and Q_P at time t . We also define $Q(t)$ to be the set of jobs pending in the algorithm at time t , where it holds that $Q(t) = Q_F(t) \cup Q_P(t)$.

In addition, we use $\delta(t)$, $\delta_F(t)$, $\delta_P(t)$ to denote the cardinalities of $Q(t)$, $Q_F(t)$, $Q_P(t)$ respectively.

We also use π_F^t and π_P^t to denote the values of π_F and π_P at time t , respectively.

Definition B.2. For every time t and job $q \in Q_F(t)$, define $X(q, t) = \{q' \in Q_F(t) \mid \pi_F(q') \leq \pi_F(q)\}$. For ease of notation, we denote $w(X(q, t))$ by $\beta(q, t)$. We also similarly define $X(q, t)$ and $\beta(q, t)$ for $q \in Q_P(t)$ according to π_P .

Definition B.3 (volume of job covered by bar). For any $x \in \mathbb{R}^+ \cup \{0\}$ and time t , and for any job $q \in Q(t)$, we define

$$\gamma_q(x, t) = \begin{cases} y_q(t) & x \geq \beta(q, t) \\ 0 & \text{otherwise} \end{cases}$$

called the volume of q covered by x at t .

Definition B.4 (volume covered by bar). We define:

1. The covered volume of x at t , which is $B(x, t) = \sum_{q \in Q(t)} \gamma_q(x, t)$.
2. $B_F(x, t) = \sum_{q \in Q_F(t)} \gamma_q(x, t)$.
3. $B_P(x, t) = \sum_{q \in Q_P(t)} \gamma_q(x, t)$.

Observe that $B(x, t) = B_F(x, t) + B_P(x, t)$

In the following analysis we consider various properties of the functions of our algorithm. When considering a function call at time t , we use t^- to denote the time immediately before the function call, and use t to denote the time immediately after the function call.

Proposition B.5. *Let t be any time in which UPONHEAVYF is called. Then for every $x \in \mathbb{R}^+ \cup \{0\}$ we have that*

$$B(x, t) = B(x, t^-)$$

Proof. Let q be the job moved from F to P . It holds that $\pi_F^{t^-}(q) = \delta_F(t^-)$ and that $\pi_P^t(q) = \delta_P(t)$. Thus, we have that for every job $q' \in Q(t) \setminus \{q\}$ it holds that $\beta(q', t) = \beta(q', t^-)$, and thus an identical amount of the volume of q' is covered by x at t^- and at t .

As for the volume of q , it holds that $\beta(q, t^-) = \delta_F(t^-)$. Now observe that $\delta_F(t^-) = \delta_P(t^-) + 1$, since UPONHEAVYF was called (it cannot be that $\delta_F(t^-) > \delta_P(t^-) + 1$, since an earlier call to UPONHEAVYF would fix this imbalance). Thus, after moving q we have that $\delta_P(t) = \delta_F(t^-)$. We therefore have that

$$\beta(q, t) = \delta_P(t) = \delta_F(t^-) = \beta(q, t^-)$$

Overall, we have that $\beta(q', t^-) = \beta(q', t)$ for every $q' \in Q(t)$, and thus $B(x, t) = B(x, t^-)$, as required. \square

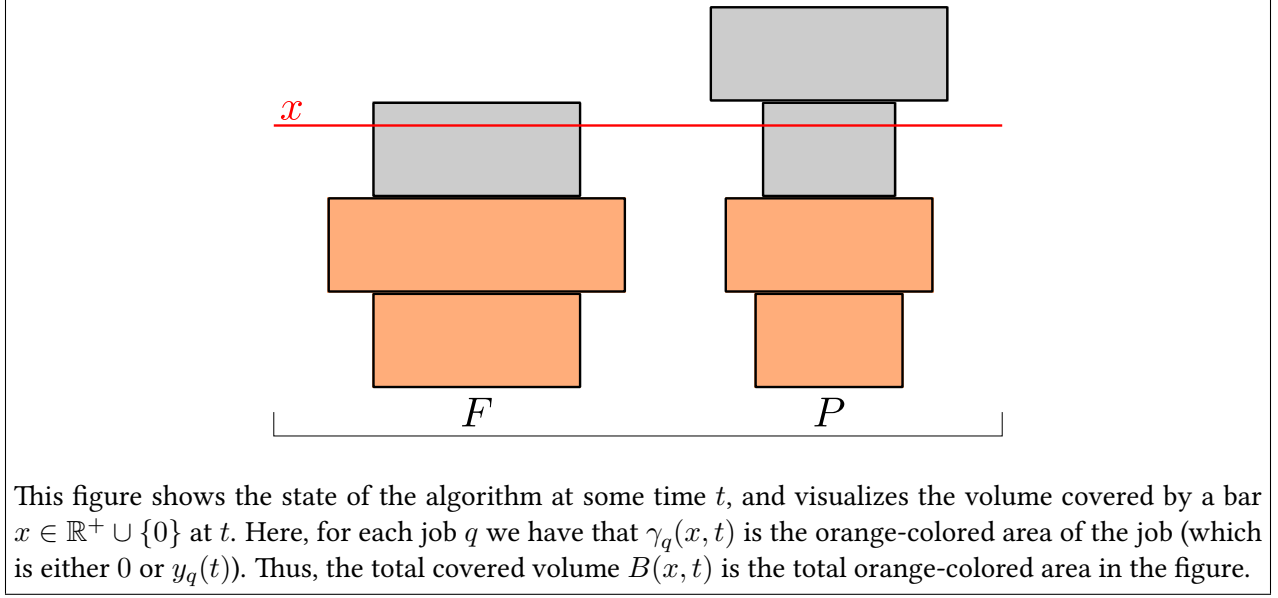


Figure 5: Volume Covered by Bar

Proposition B.6. Consider a call to `UPONJOBRELEASE(q)` at time t . For every $x \in \mathbb{R}^+ \cup \{0\}$, it holds that

$$B(x, t) \geq B(x, t^-)$$

Proof. We denote by t_0 the “time” immediately after adding q to π_F – that is, the state of the algorithm before Line 3 in `UPONJOBRELEASE(q)`.

Claim 1: $B(x, t_0) \geq B(x, t^-)$. Since $\pi_F^{t_0}(q) = \delta_F(t_0)$, we have that $\pi_F^{t_0}(q) > \pi_F^{t_0}(q')$ for every $q' \in Q_F(t^-)$. For every such q' we thus have $\beta(q', t_0) = \beta(q', t^-)$, and therefore $B(x, t_0) \geq B(x, t^-)$.

Claim 2: $B(x, t) \geq B(x, t_0)$. If no violations exist at t_0 , and thus no rotation is performed, then the claim holds. Otherwise, the cyclic permutation $(\pi_F(q_m), \pi_F(q_{m-1}), \dots, \pi_F(q_1), \pi_F(q))$ is applied, where the jobs q_1, \dots, q_m are the jobs involved in violations with q at t_0 , such that $\pi_F^{t_0}(q_i) > \pi_F^{t_0}(q_{i-1})$ for every $i > 0$.

We write $q_0 = q$, and observe the set of jobs $Q' = \{q_0, q_1, \dots, q_m\}$ involved in the cyclic permutation. Since the only change is in priorities between t and t_0 is in Q' , it remains to observe the change in the volume of Q' covered by x . If $\pi_F^{t_0}(q') > x$ for every $q' \in Q'$, the volume of Q' covered by x does not change. The same holds if $\pi_F^{t_0}(q') \leq x$ for every $q' \in Q'$.

It remains to observe the case in which x covers some of Q' but not all. In this case, let i be the smallest index such that $\pi_F^{t_0}(q_i) \leq x$, such that $i > 0$. After applying the cyclic permutation, we have that the volume covered by x has decreased by p_{q_i} (since $\pi_F^t(q_i) = \pi_F^{t_0}(q_{i-1})$), but has increased by p_{q_0} (since $\pi_F^t(q_0) = \pi_F^{t_0}(q_m)$). Now, recall that $q_0 = q$, and that (q, q_i) was a violation at t_0 . This implies that

$$p_{q_i} < \mu \tilde{p}_{q_i} \leq \tilde{p}_q \leq p_q$$

proving the second claim.

Combining the two claims, we have that

$$B(x, t) \geq B(x, t^-)$$

as required. □

Proposition B.7. *Outside of function calls, there are no violations in the algorithm.*

Proof. Observe that the only changes to F occur in calls to UPONJOBRELEASE and UPONHEAVYF. We show that if there is no violation before such a function call, then there would be no violation after the function call.

In the case of UPONHEAVYF, this trivially holds, since removing a job from F cannot cause a new violation. It remains to consider UPONJOBRELEASE.

Suppose that UPONJOBRELEASE(q) is called at time t . As in the proof of Proposition B.6, we denote by t_0 the time immediately after the insertion of the job q into F , and before the possible rotation of Line 3. By the induction hypothesis, and through the fact that $\pi_F^{t_0}(q) = \delta_F(t_0)$, the only violations at t_0 are of the form (q, q') for some $q' \in Q_F(t_0) \setminus \{q\}$. If there are no such violations then no rotation occurs, and thus the state at time t contains no violations, completing the proof.

Otherwise, denoting $q_0 = q$, the set $Q' = \{q_0, q_1, \dots, q_m\}$ is rotated. For ease of notation, we write $Q'' = Q_F(t_0) \setminus Q'$. Consider the possible violations at time t , after this rotation.

1. **violations of the form (q''_1, q''_2) for some $q''_1, q''_2 \in Q''$.** Such violations cannot occur in t , since they did not exist in t_0 , and since $\pi_F^t(q'') = \pi_F^{t_0}(q'')$ for every $q'' \in Q''$.
2. **violations between q_i and q'' for some index $0 < i \leq m$ and some job $q'' \in Q''$.** Consider that there was no violation between q_i and q'' at time t_0 . Since $\pi_F^t(q'') = \pi_F^{t_0}(q'')$, and since $\pi_F^t(q_i) \geq \pi_F^{t_0}(q_i)$, it must be that the violation is of the form (q_i, q'') .
Now, observe that $\pi_F^t(q_i) \leq \pi_F^t(q)$. In addition, since (q, q_i) was a violation at t_0 , it holds that $\tilde{p}_{q_i} < \mu \tilde{p}_{q_i} \leq \tilde{p}_q$. Thus, if (q_i, q'') is a violation at t , it must be that (q, q'') was a violation at t_0 , in contradiction to $q'' \in Q''$.
3. **violations of the form (q_i, q_j) for $1 \leq i, j \leq m$.** Observe that there was no violation between q_i and q_j at time t_0 , and that the order between their priorities did not change between t_0 and t . Thus, there is no violation between them at t .
4. **violations between q_i and q , for some index i .** Consider that since $\pi_F^t(q) \leq \pi_F^t(q_i)$, the violation must be of the form (q_i, q) . Now, consider that (q, q_i) was a violation at t_0 , and thus $\mu \tilde{p}_q > \tilde{p}_q \geq (1 + \epsilon_{q_i}) \tilde{p}_{q_i} > \tilde{p}_{q_i}$. Thus, (q_i, q) cannot be a violation at t .
5. **violations between q and q'' , for some $q'' \in Q''$.** Since (q, q'') was not a violation at t_0 , and since $\pi_F^t(q'') = \pi_F^{t_0}(q'')$ and $\pi_F^t(q) \leq \pi_F^{t_0}(q)$, we have that (q, q'') is not a violation at t . It remains to consider the possible violation (q'', q) .
Now, consider that $\pi_F^t(q) = \pi_F^{t_0}(q_m)$. In addition, since (q, q_m) was a violation at t_0 , it holds that $\mu \tilde{p}_{q_m} \leq \tilde{p}_q \leq \mu \tilde{p}_q$. Thus, if (q'', q) is a violation at t , it must be that (q'', q_m) was a violation at t_0 , in contradiction to the fact that all violations at t_0 involve q .

Overall, there are no violations at time t . □

For ease of notation, we define $\theta = \lceil \mu^2 \rceil$.

Proposition B.8. *Consider a call to UPONJOBRELEASE(q) at time t . For every $x \in \mathbb{R}^+ \cup \{0\}$, it holds that*

$$B(x + \theta, t) \geq B(x, t^-) + p_q$$

Proof. As in the proofs of Propositions B.6 and B.7 Propositions B.6 and B.8, we denote by t_0 the time immediately after q 's insertion into F , and before the possible rotation. Consider that for ev-

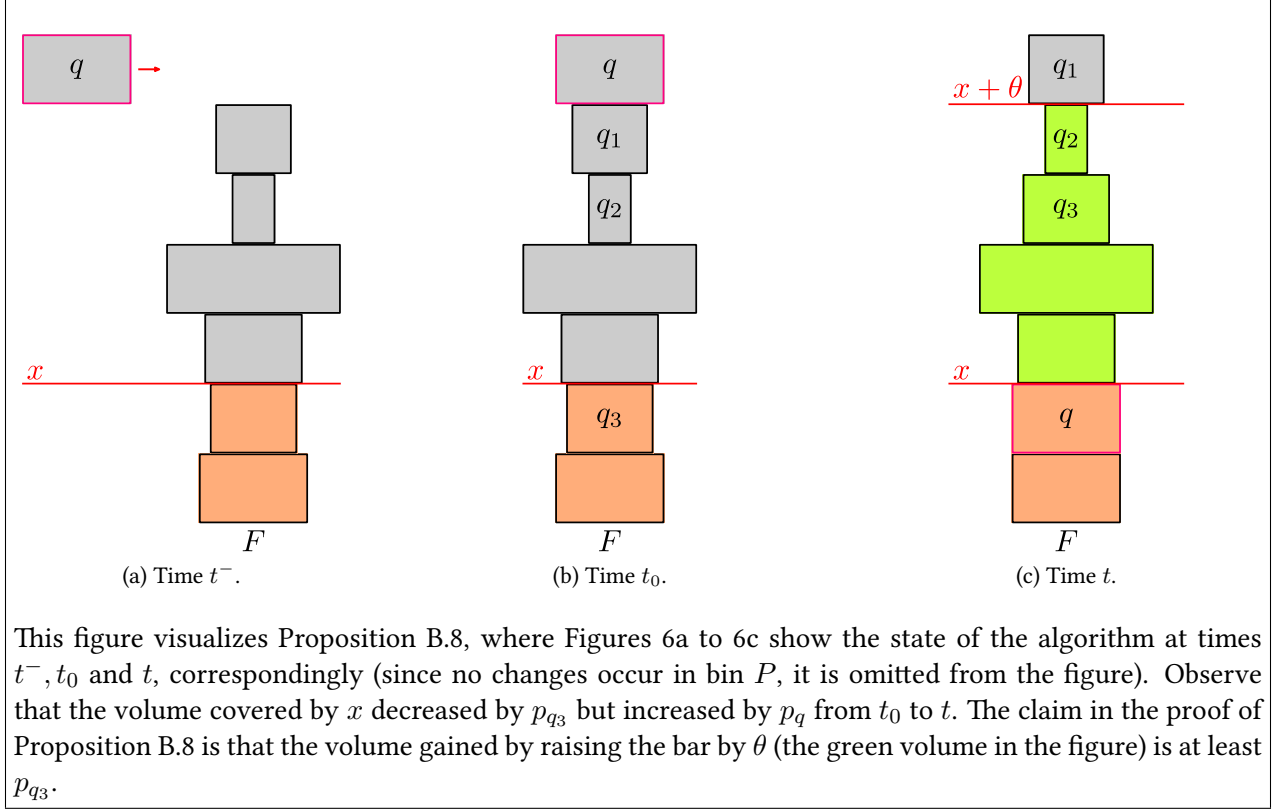


Figure 6: Visualization of Proposition B.8

ery $q' \in Q(t) \setminus \{q\}$ it holds that $\beta(q', t_0) = \beta(q', t^-)$, and thus $\gamma_{q'}(x, t_0) = \gamma_{q'}(x, t^-)$, which implies that $B(x, t_0) \geq B(x, t^-)$.

If, in addition to the previous observation, it holds that $x \geq \pi_F^{t_0}(q)$, then we have that $\gamma_q(x, t_0) = p_q$, and thus $B(x, t_0) \geq B(x, t^-) + p_q$. We thus have that

$$B(x + \theta, t) \geq B(x, t) \geq B(x, t_0) \geq B(x, t^-) + p_q$$

where the second inequality uses Claim 2 of the proof of Proposition B.6. This completes the proof for the case that $x \geq \pi_F^{t_0}(q)$, and we thus assume for the remainder of the proof that $x < \pi_F^{t_0}(q)$.

Denote $q_0 = q$, and let q_1, \dots, q_m be the jobs involved in violations with q at t_0 (it is possible that $m = 0$ if there are no violations). The algorithm performs applies the cyclic permutation $(\pi_F^{t_0}(q_m), \dots, \pi_F^{t_0}(q_1), \pi_F^{t_0}(q_0))$ to $\pi_F^{t_0}$ to obtain π_F^t .

As in the proof of Proposition B.6, let i be the minimal index such that $\pi_F^{t_0}(q_i) \leq x$ (due to the assumption that $\pi_F^{t_0}(q) > x$, we have that $i > 0$). Due to the rotation, the volume covered by x has decreased by p_{q_i} but increased by p_{q_0} . That is, we have that

$$B(x, t) \geq B(x, t_0) - p_{q_i} + p_{q_0}$$

Now, it remains to show that

$$B(x + \theta, t) \geq B(x, t) + p_{q_i} \tag{2}$$

which completes the proof.

We henceforth write q^* instead of q_i . It holds that $x < \beta(q^*, t)$, and thus $\gamma_{q^*}(x, t) = 0$. If, in addition, we have that $\beta(q^*, t) \leq x + \theta$, then $\gamma_{q^*}(x + \theta, t) = p_{q^*}$, which implies Equation (2) and completes the proof.

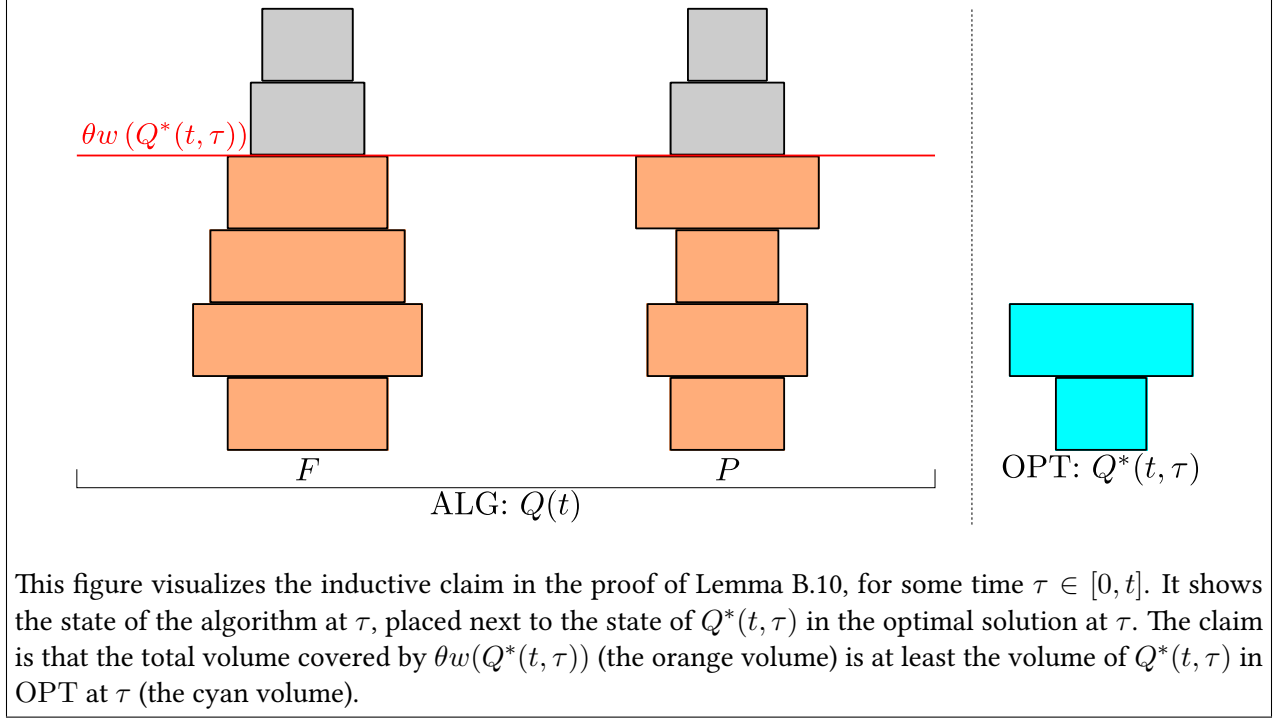


Figure 7: Lemma B.10

Otherwise, we have that $x + \theta < \beta(q^*, t)$. This implies that there exist θ jobs $q'_1, \dots, q'_\theta \in Q_F(t)$, such that $\beta(q'_j, t) = \lfloor x + j \rfloor$ for every index $1 \leq j \leq \theta$. Observe that for each index j it holds that $\gamma_{q'_j}(x, t) = 0$ and $\gamma_{q'_j}(x + \theta, t) = p_{q'_j}$. Thus, it holds that

$$B(x + \theta, t) \geq B(x, t) + \sum_{j=1}^{\theta} p_{q'_j} \quad (3)$$

It holds that $\pi_F^t(q'_j) \leq \pi_F^t(q^*)$ for each index j . Using Proposition B.7, we have

$$p_{q'_j} \geq \tilde{p}_{q'_j} \geq \frac{\tilde{p}_{q^*}}{\mu} > \frac{p_{q^*}}{\mu^2} \geq \frac{p_{q^*}}{\theta}$$

Plugging into Equation (3) yields Equation (2), completing the proof. \square

Proposition B.9. For any x and time t , if $B(x, t)$ is decreasing due to the algorithm's processing of a job, then $B(x, t) = V(t)$.

Proof. Suppose that $B(x, t)$ decreased due to the processing of a job $q \in Q_P(t)$ by the algorithm. This implies that $\gamma_q(x, t) > 0$, and thus $x \geq \beta(q, t)$. But from the choice of job in PROCESS, it holds that $\pi_P^t(q) = \delta_P(t)$, and thus $\beta(q, t) = \delta_P(t)$. Therefore, for every $q' \in Q_P(t)$ it holds that $\gamma_{q'}(x, t) = y_{q'}(t)$. Moreover, since the algorithm maintains that $\delta_F(t) \leq \delta_P(t)$, for every $q' \in Q_F(t)$ we also have that $\gamma_{q'}(x, t) = y_{q'}(t)$.

Overall, we have that $B(x, t) = V(t)$, as required. \square

Lemma B.10. At any time t it holds that $B(\theta\delta^*(t), t) = V(t)$.

Proof. For any time τ , let $Q^*(\tau)$ be the set of jobs pending in the optimal solution at time τ . We now fix a time t . Let τ be any time in the range $[0, t]$. We also define $Q^*(t, \tau) = Q^*(t) \cap Q^*(\tau)$ – i.e. the jobs alive at t which were already released by τ . For ease of notation we write $Y^*(\tau) = \sum_{q \in Q^*(t, \tau)} y_q^*(\tau)$.

We prove, by induction on τ , that for every $\tau \in [0, t]$ it holds that

$$B(\theta \cdot |Q^*(t, \tau)|, \tau) \geq Y^*(\tau) \quad (4)$$

A visualization of the claim in Equation (4) appears in Figure 7.

Clearly, Equation (4) holds for $\tau = 0$, as $Y^*(\tau) = 0$. Now, we show that no possible event can break the inequality of Equation (4) as time τ progresses. When considering an event at τ , we denote by τ^- the time immediately before the event. Consider the possible events:

1. **A job moves from F to P in the algorithm.** Proposition B.5 implies that the left-hand side of Equation (4) does not decrease upon this event. Since the right-hand side does not change, the inequality continues to hold.
2. **A job $q \notin Q^*(t)$ is released.** In this case, the right-hand side of Equation (4) remains the same. Proposition B.6 implies that the left-hand side does not decrease.
3. **A job $q \in Q^*(t)$ is released.** In this case, the right-hand side of Equation (4) increases by p_q , as $Q^*(t, \tau^-) = Q^*(t, \tau) \cup \{q\}$. Proposition B.8 implies that

$$B(\theta \cdot |Q^*(t, \tau)|, \tau) = B(\theta \cdot |Q^*(t, \tau^-)| + \theta, \tau) \geq B(\theta \cdot |Q^*(t, \tau^-)|, \tau^-) + p_q$$

and thus the inequality of Equation (4) continues to hold.

4. **A job $q \in Q(\tau)$ is processed.** Proposition B.9 implies that if the left-hand side decreases as a result of processing, then $B(\theta \cdot |Q^*(t, \tau)|, \tau) = V(\tau)$. Since the algorithm is not lazy (i.e. always processes a pending job if there exists one), it holds that $V(\tau) = V^*(\tau) \geq Y^*(\tau)$, and thus the inequality holds.

The proof of the induction claim is complete. Now observe that choosing $\tau = t$, we have that $|Q^*(t, \tau)| = \delta^*(t)$ and that $Y^*(t) = V^*(t) = V(t)$. Thus, Equation (4) yields that

$$B(\theta \delta^*(t), t) = V(t)$$

completing the proof. □

Proof of Theorem 4.2. Lemma B.10 implies that at any time t we have $B(\theta \delta^*(t), t) = V(t)$. This implies that $\delta_F(t) \leq \theta \delta^*(t)$; otherwise, there would exist a pending job $q \in Q_F(t)$ such that $\beta(q, t) > \theta \delta^*(t)$, the volume of which would not be counted in $B(\theta \delta^*(t), t)$. The same argument applies to P as well, yielding that $\delta_P(t) \leq \theta \delta^*(t)$.

Overall, we get that $\delta(t) \leq 2\theta \delta^*(t)$. Integrating over t completes the proof of the theorem. □

B.2 Proof of Theorem 4.3

In the following analysis, we prove Theorem 4.3.

Definition B.11 (time-dependent variable values). For any time t , we define $Q_{Fi}(t)$ and $Q_{Pi}(t)$ to be the values of the variables Q_{Fi} and Q_{Pi} at time t , respectively. We define $Q_{Ai}(t) = Q_{Fi}(t) \cup Q_{Pi}(t)$,

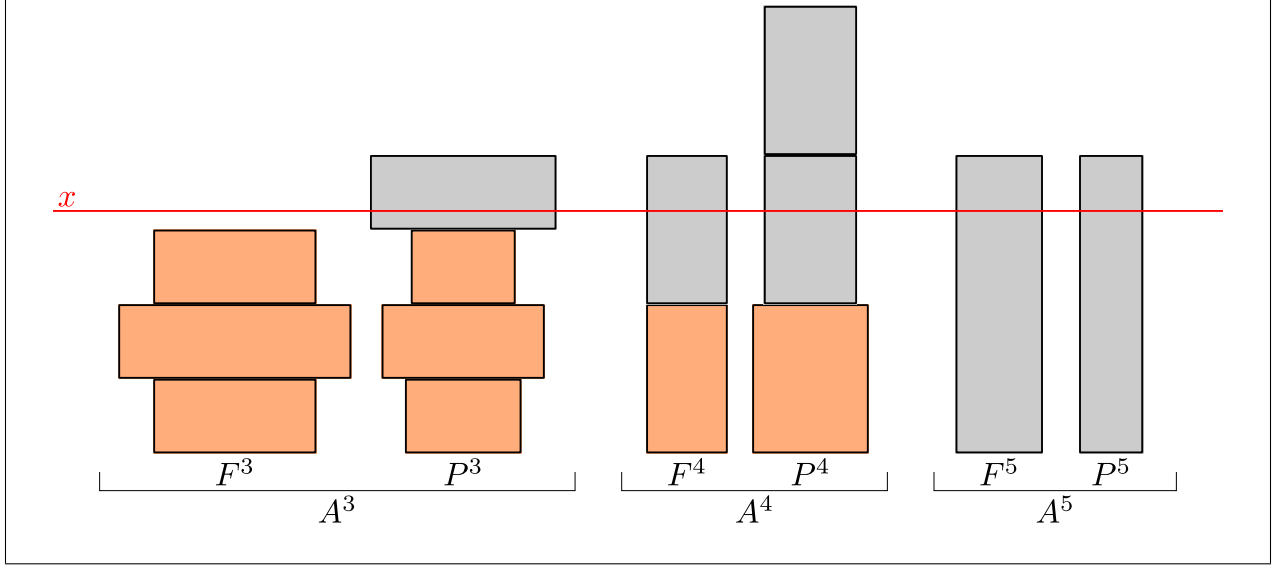


Figure 8: Visualization of Covered Volume

which is the set of pending jobs in superbin A^i at time t .

We also define $\pi_{F^i}^t$ and $\pi_{P^i}^t$ to be the values of the priority-mapping variables π_{F^i} and π_{P^i} at time t .

Definition B.12 (base of a job). For every time t and every job $q \in Q_{F^i}(t)$, we define the *base* of q at time t as follows:

$$X(q, t) = \{q' \in Q_{F^i}(t) \mid \pi_{F^i}^t(q') \leq \pi_{F^i}^t(q)\}$$

For ease of notation, we denote $w(X(q, t))$ by $\beta(q, t)$. For every job $q \in Q_{P^i}(t)$, we similarly define $X(q, t)$ and $\beta(q, t)$ according to $\pi_{P^i}^t$.

We now redefine the volume covered by a bar x .

Definition B.13 (Volume covered by bar). For any $x \in \mathbb{R}^+ \cup \{0\}$ and time t , and for any job $q \in Q(t)$, we define

$$\gamma_q(x, t) = \begin{cases} y_q(t) & x \geq \beta(q, t) \\ 0 & \text{otherwise} \end{cases}$$

called the volume of q covered by x at t .

For every superbin index i , we define the total volume covered by x in various bins:

- $B_{F^i}(x, t) = \sum_{q \in Q_{F^i}(t)} \gamma_q(x, t)$
- $B_{P^i}(x, t) = \sum_{q \in Q_{P^i}(t)} \gamma_q(x, t)$
- $B_{A^i}(x, t) = B_{F^i}(x, t) + B_{P^i}(x, t)$
- $B(x, t) = \sum_i B_{A^i}(x, t)$

Figure 8 visualizes Definition B.13, where $B(x, t)$ is the total orange area in the figure.

For ease of notation, we again define $\theta = \lceil (1 + \epsilon)^2 \rceil$.

Proposition B.14. Upon the release of a job q at time t , and for every $x \in \mathbb{R}^+ \cup \{0\}$, it holds that

- $B(x, t) \geq B(x, t^-)$

- $B(x + \theta w_q, t) \geq B(x, t^-) + p_q$

Proof. Let i be such that $w_q = 2^i$. The job q is thus sent to the superbin A^i .

To prove the first claim, first note that the structure of every superbin $A^{i'}$ for $i' \neq i$ remains the same after q 's release. Thus, we have that $B_{A^{i'}}(x, t) = B_{A^{i'}}(x, t^-)$. As for the volume covered in A^i , we know that it does not decrease due to Proposition B.6 (it is easy to verify that this proposition holds when all weights are scaled up by 2^i).

To prove the second item, again note that for every $i' \neq i$ it holds that $B_{A^{i'}}(x + \theta w_q, t) \geq B_{A^{i'}}(x, t) = B_{A^{i'}}(x, t^-)$. As for the volume covered in A^i , using Proposition B.8 (scaled up by 2^i) yields that $B_{A^i}(x + \theta w_q, t) \geq B_{A^i}(x, t^-) + p_q$. This completes the proof. \square

Proposition B.15. *For any x and time t , if $B(x, t)$ is decreasing due to the algorithm's processing of a job, then $B(x, t) = V(t)$.*

Proof. Consider the job q being processed, where $q \in P^i$ for some i . Since $\gamma_q(x, t) > 0$, we have that $x \geq \beta(q, t)$. Since the algorithm processes the job $q \in Q_{P^i}(t)$ such that $\pi_{P^i}^t(q) = \delta_{P^i}(t)$, we have that $\beta(q, t) = \delta_{P^i}(t) \cdot 2^i = w(Q_{P^i}(t))$.

Now, observe that since the algorithm chose superbin A^i for processing, it must be that for every i' we have $w(Q_P^i(t)) \geq w(Q_P^{i'}(t)) \geq w(Q_F^{i'}(t))$, where the second inequality is since the algorithm moves jobs from the full bins to the partial bins.

Since x is at least the weight of every bin in the system, it is at least $\beta(q', t)$ for every $q' \in Q(t)$, and thus $B(x, t) = V(t)$. \square

Lemma B.16. *At any time t it holds that $B(\theta \cdot w(Q^*(t)), t) = V(t)$.*

Proof. The proof of this lemma is nearly identical to that of Lemma B.10 of Subsection 4.1, but uses the propositions of this subsection instead.

We now fix a time t . Let τ be any time in the range $[0, t]$. We again define $Q^*(t, \tau)$ and $Y^*(\tau)$ as in the proof of Lemma B.10.

We prove, by induction on τ , an identical claim to that in the proof of Lemma B.10, which is that for every $\tau \in [0, t]$ it holds that

$$B(\theta \cdot |Q^*(t, \tau)|, \tau) \geq Y^*(\tau) \quad (5)$$

The claim trivially holds for time $\tau = 0$, before any requests were released. We now consider the possible events which could possibly break the claim as τ progresses, in a similar way to Lemma B.10.

1. **A job moves from F to P in the algorithm.** Proposition B.5 implies that the left-hand side of Equation (5) does not decrease upon this event. Since the right-hand side does not change, the inequality continues to hold.
2. **A job $q \notin Q^*(t)$ is released.** In this case, the right-hand side of Equation (5) remains the same. The first claim in Proposition B.14 implies that the left-hand side does not decrease.
3. **A job $q \in Q^*(t)$ is released.** In this case, the right-hand side of Equation (5) increases by p_q , as $Q^*(t, \tau^-) = Q^*(t, \tau) \cup \{q\}$. The second claim in Proposition B.14 implies that

$$B(\theta \cdot w(Q^*(t, \tau)), \tau) = B(\theta \cdot w(Q^*(t, \tau^-)) + \theta w_q, \tau) \geq B(\theta \cdot w(Q^*(t, \tau^-)), \tau^-) + p_q$$

and thus the inequality of Equation (5) continues to hold.

4. **A job $q \in Q(\tau)$ is processed.** Proposition B.15 implies that if the left-hand side decreases as a result of processing, then $B(\theta \cdot |Q^*(t, \tau)|, \tau) = V(\tau)$. Since the algorithm is not lazy (i.e. always

processes a pending job if there exists one), it holds that $V(\tau) = V^*(\tau) \geq Y^*(\tau)$, and thus the inequality holds.

The proof of the induction claim is now complete, and choosing $\tau = t$ in Equation (5) yields

$$B(\theta\delta^*(t), t) = V(t)$$

as required. \square

Proof of Theorem 4.3. At any time t , the algorithm contains at most $\lceil \log W \rceil + 1$ nonempty superbins, each of which contains two bins (full and partial). From Lemma B.16, at any time t , there does not exist a pending job q in the algorithm such that $\beta(q, t) > \theta \cdot w(Q^*(t))$. Thus, the total weight in each bin is at most $\theta \cdot w(Q^*(t))$, and thus $2\theta \cdot w(Q^*(t))$ for each superbin (which contains two bins). Overall, the total weight in the system is at most $2\theta(\lceil \log W \rceil + 1)w(Q^*(t))$.

Integrating over t thus yields the desired result, which is that the algorithm is $O(\theta \log W)$ competitive. \square

C Lower Bound for Unweighted SPPT

In this section, we show a lower bound of 2 on the competitive ratio of any algorithm for SPPT, even when the distortion parameter is arbitrarily close to 1. This lower bound is perhaps somewhat surprising, as one would hope that as μ approaches 1, one would be able to approximate the optimal SRPT schedule, and thus approach 1-competitiveness.

Theorem C.1. *For any $\mu > 1$, there is no deterministic μ -robust algorithm with competitive ratio less than 2.*

Proof. Fix some $\mu > 1$. We assume that $\mu \leq 2$, as we are interested in the case of μ approaching 1 (clearly if the lower bound holds for small μ , it holds for large μ).

Let $\lambda = \frac{\mu+1}{\mu-1}$, and let M be an arbitrarily large integer. The adversary we describe operates in M phases, numbered from $M - 1$ down to 0. Phase i takes λ^i time units. Observe the following adversary:

1. for i from $M - 1$ to 0:
 - (a) release two jobs of predicted processing time λ^i .
 - (b) wait λ^i time units.
 - (c) of the two jobs released in this phase, let q_1^i be the job processed more by the algorithm during these λ^i time units, and let q_2^i be the other job. The adversary sets $p_{q_1^i} = \mu\lambda^i$ and $p_{q_2^i} = \lambda^i$.

During phase i , the algorithm has processed job q_1^i for at most $\frac{\lambda^i}{2}$ time, and has processed job q_2^i for at most λ^i time. Thus, both q_1^i and q_2^i have remaining volume of $(\mu - 1)\lambda^i = (\mu + 1)\lambda^{i-1}$ at the end of the phase. Also observe that the total time of phases $i - 1$ through 0 is at most

$$\sum_{i'=0}^{\infty} \lambda^{i-1-i'} = \lambda^{i-1} \cdot \frac{1}{1 - \lambda^{-1}} = \lambda^{i-1} \cdot \frac{1 - \mu}{2}$$

And thus, at the end of the last phase, it must be that the remaining processing time for both q_1^i and q_2^i in the algorithm is at least $\frac{1-\mu}{2}\lambda^{i-1}$.

Overall, we have that the algorithm has $2M$ living jobs at the end of the M phases, each with at least $\frac{1-\mu}{2}\lambda^0 = \frac{1-\mu}{2}$ remaining volume. The optimal solution, on the other hand, has M living jobs, as it would complete q_2^i in each phase i .

Now, after the M phases, the adversary would begin the “bombardment”, releasing a job of processing time $x = \frac{1-\mu}{2}$ every x time units. The algorithm would have no option which is better than serving the “bombardment” requests, and the optimal solution would do the same. During the bombardment, the algorithm has $2M+1$ living jobs at any time, and the optimal solution has $M+1$. Thus, the competitive ratio tends to $\frac{2M+1}{M+1}$ as the bombardment continues. Since M was chosen arbitrarily, we can let M tend to ∞ , and thus the competitive ratio tends to 2. \square

D Semiclairvoyant Scheduling

In the semicclairvoyant model, we are given the logarithmic class of a job rather than an estimate for its processing time. That is, for a job q the algorithm is given $\ell_q = \lfloor \log_\rho p_q \rfloor$ and not p_q (for a constant $\rho > 1$). Clearly, this model can be reduced to the prediction model as follows. Upon the arrival of q , define $\tilde{p}_q = \rho^{\ell_q}$ and $\mu = \rho$. From the definition of class, it is clear that indeed $p_q \in [\tilde{p}_q, \mu\tilde{p}_q]$.

Denote the semicclairvoyant model with the parameter ρ by SC_ρ . Applying the theorems of this paper for $SPPT$, we immediately obtain the first results for weighted flow time in the semicclairvoyant setting.

Corollary D.1 (of Theorems 3.1, 3.11 and 4.3). *In the SC_ρ model, there exist:*

1. An $O(\log P)$ -competitive algorithm (using Theorem 3.1)
2. An $O(\log D)$ -competitive algorithm (using Theorem 3.11)
3. An $O(\log W)$ -competitive algorithm (using Theorem 4.3)

Note that these results for the semicclairvoyant setting match the best known results for the clairvoyant setting, in terms of all three parameters P, D, W .

In addition, we consider the algorithm in Subsection 4.1 for unweighted $SPPT$, and show that for the SC_ρ model it obtains an improved competitive ratio, which is $2\lceil \rho \rceil$. In particular, when $\rho = 2$ this algorithm is 4 competitive, improving upon the 13-competitive algorithm of [6].

D.1 Unweighted Semicclairvoyant Scheduling

We now show the following theorem for the competitiveness of the algorithm of Subsection 4.1 for the unweighted semicclairvoyant setting.

Theorem D.2. *In the SC_ρ model, the competitive ratio of the algorithm of Subsection 4.1 is $2\lceil \rho \rceil$.*

Corollary D.3. *In the SC_2 model (i.e. base-2 classes) the algorithm of Subsection 4.1 is 4-competitive.*

As observed before, we have that $\mu = \rho$.

For ease of notation, we define $\theta' = \lceil \mu \rceil = \lceil \rho \rceil$. The proof of Theorem D.2 is identical to that of Theorem 4.2, except for replacing Proposition B.8 with the following proposition.

Proposition D.4 (stronger version of Proposition B.8). *Consider a call to $UPONJOBRELEASE(q)$ at time t . For every $x \in \mathbb{R}^+ \cup \{0\}$, it holds that*

$$B(x + \theta', t) \geq B(x, t^-) + p_q$$

Proof. The proof is identical to the proof of Proposition B.8, except for replacing the claim that

$$B(x + \theta, t) \geq B(x, t) + p_{q'}$$

with the stronger claim that

$$B(x + \theta', t) \geq B(x, t) + p_{q'}$$

with q' defined as in the proof of Proposition B.8.

From the definition of q' in the original proof, it holds that $\beta(q', t) > x$, and thus $\gamma_{q'}(x, t) = 0$. If $\beta(q', t) \leq x + \theta'$, then $\gamma_{q'}(x + \theta', t) = p_{q'}$, and the proof is complete.

Otherwise, it holds that $\beta(q', t) > x + \theta'$. Consider the θ' distinct jobs $q_1, \dots, q_{\theta'} \in Q_F(t)$ such that $\beta(q_j, t) = x + j$ for each index j . Observe that for each index j it holds that $\gamma_{q_j}(x, t) = 0$ and $\gamma_{q_j}(x + \theta', t) = p_{q_j}$. Thus, it holds that

$$B(x + \theta', t) \geq B(x, t) + \sum_{j=1}^{\theta'} p_{q_j} \quad (6)$$

It holds that $\pi_F^t(q') > \pi_F^t(q_j)$ for each j . Using Proposition B.7, we have that the state of the algorithm at time t contains no violations. However, observe that if $\ell_{q_j} < \ell_{q'}$, then we have that

$$\rho \tilde{p}_{q_j} = \rho \cdot \rho^{\ell_{q_j}} = \rho^{\ell_{q_j} + 1} \leq \rho^{\ell_{q'}} = \tilde{p}_{q'}$$

which is a violation, in contradiction to Proposition B.7. Thus, for every j it holds that $\ell_{q_j} \geq \ell_{q'}$, and thus $p_{q_j} \geq \tilde{p}_{q_j} \geq \rho^{\ell_{q_j}} \geq \rho^{\ell_{q'}} \geq \frac{p_{q'}}{\rho} \geq \frac{p_{q'}}{\theta'}$.

Plugging into Equation (6), we get

$$B(x + \theta', t) \geq B(x, t) + p_{q'}$$

completing the proof. □

This completes the proof of Theorem D.2.

E Proof of Theorem 3.11

This section proves Theorem 3.11.

Observe that the maximum ratio of *estimated* densities (i.e. $\frac{w_q}{\tilde{p}_q}$) with respect to the original weight (before rounding) is at most μD . After rounding up to powers of λ , which is $\Theta(\mu)$, we have that the maximum ratio of densities becomes $O(\mu^2 D)$. Hence, the maximum ratio of estimated EI-density values is also $O(\mu^2 D)$. We thus define the number of EI-density classes to be Λ' , and observe that $\Lambda' = O(\log(\mu^2 D)) = O(\log(\mu D))$.

The following lemma immediately implies Theorem 3.11, in the same manner in which Lemma 3.4 implied Theorem 3.1.

Lemma E.1 (analogue of Lemma 3.4). *At any point in time t , it holds that $W(t) \leq O(\Lambda' \mu^2) \cdot W^*(t)$.*

Proof of Lemma E.1. As noted in the proof of Lemma 3.4, we have $\text{clw}(t) \leq i^*$. Define j_{\min} and j_{\max} to

be the minimum and maximum EI-density classes, respectively. We have

$$\begin{aligned}
W(t) &= \sum_{j=j_{\min}}^{j_{\max}} W_{\leq i^*, =j}(t) \\
&\leq 2(j_{\max} - j_{\min} + 1)u_{i^*} + 2\mu W^*(t) + \sum_{j=j_{\min}}^{j_{\max}} 2 \max\left\{0, \frac{\Delta V_{\leq i^*, \leq j}(t)}{2^j}\right\} \\
&\leq 2\Lambda' u_{i^*} + 2\mu W^*(t) + \sum_{j=j_{\min}}^{j_{\max}} 8\mu \cdot u_{\text{clw}(t_j)}
\end{aligned}$$

using the same arguments as in the proof of Lemma 3.4. Now, observe that for every EI-density class j we have $\text{clw}(t_j) \leq i^*$ from Proposition 3.10. Thus

$$W(t) \leq (8\mu + 2)\Lambda' u_{i^*} + 2\mu W^*(t) \leq O(\Lambda' \mu^2) W^*(t) \quad \square$$