Distortion-Oblivious Algorithms for Minimizing Flow Time

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

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

Noam Touitou
noam.touitou@cs.tau.ac.il
Tel Aviv University

Abstract

We consider the classic online problem of scheduling on a single machine to minimize total flow time. In STOC 2021, the concept of robustness to distortion in processing times was introduced: for every distortion factor μ, an $O(\mu^2)$-competitive algorithm ALG_μ which handles distortions up to μ was presented. However, using that result requires one to know the distortion of the input in advance, which is impractical.

We present the first *distortion-oblivious* algorithms: algorithms which are competitive for *every* input of *every* distortion, and thus do not require knowledge of the distortion in advance. Moreover, the competitive ratios of our algorithms are $\tilde{O}(\mu)$, which is a quadratic improvement over the algorithm from STOC 2021, and is nearly optimal (we show a randomized lower bound of $\Omega(\mu)$ on competitiveness).

1 Introduction

We consider the classic online scheduling problem on a single machine. Jobs arrive over time, and the completion of each job requires some processing time on the machine. The goal of the algorithm is to minimize the *total flow time*, which is the sum over jobs of their time pending (i.e. from release to completion). This is an online problem, as the algorithm must decide which job to process at any given time, and is unaware of jobs that have not yet been released.

The classic setting. In the classic setting, the processing time of a job becomes known to the algorithm upon the job's release. This knowledge gives great power to the algorithm: a classic result from the 1950s by Smith [22] shows that the SRPT (shortest remaining processing time) algorithm is 1-competitive for this problem.

However, the assumption that processing times can be exactly known does not usually hold in practice. For example, if a job is a computer program, the cases in which the running time is known exactly in advance are very rare.

The robust model. This lack of knowledge regarding processing times is addressed by the *robust scheduling* model, presented in [1]. In this model, upon the release of a job, the algorithm is provided an estimate for the processing time of the job. Such estimates could be realistically obtained through heuristics, machine-learning models, or user input. Naturally, one would expect the competitive ratio of an algorithm to improve with the accuracy of the provided estimations.

We define the *overestimation* of an input to be the maximum estimated-to-real ratio of a job's processing time. Similarly, we define the *underestimation* to be the maximum real-to-estimated ratio of a job's processing time. Finally, the *distortion* of an input, denoted by μ, is the product of the overestimation and the underestimation of that input[1].

One desires a competitive ratio that is a function of this distortion μ. Formally, for every distortion cap $\hat{\mu} > 1$, an $O(f(\hat{\mu}))$-competitive algorithm is $\hat{\mu}$-*robust* if it remains $O(f(\hat{\mu}))$-competitive for all inputs in which $\mu \leq \hat{\mu}$ (i.e. the distortion does not exceed the distortion cap).

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

†Supported by ERC Advanced Grant "Algorithmic and Mechanism Design Research for Online Markets" (AMDROMA), MIUR PRIN 2017 Project "ALGADIMAR".

[1]The maximum ratios (and not e.g. average ratios) are indeed the correct parameters to consider; see [1].

In [1], for every distortion cap $\hat{\mu}$, a $\hat{\mu}$-robust, $O(\hat{\mu}^2)$-competitive algorithm $\text{ALG}_{\hat{\mu}}$ is presented. A major drawback of this result is that the algorithms for different values of $\hat{\mu}$ are different, which limits their usability. The two options for using these algorithms are:

- If one knows μ in advance, one can choose $\hat{\mu} = \mu$ and use ALG_μ.
- One can guess a value for $\hat{\mu}$ and hope that it is close to μ.

The first option is problematic, as knowing the distortion exactly in advance is rather unreasonable. The second option is quite perilous; we show in Appendix C the poor consequences of choosing the distortion cap $\hat{\mu}$ inaccurately:

1. When $\mu \ll \hat{\mu}$, the algorithm $\text{ALG}_{\hat{\mu}}$ is $\Omega(\hat{\mu})$-competitive, i.e. its competitiveness is not a function of the actual distortion μ but of the distortion cap $\hat{\mu}$ (this happens even when there is *no* distortion!).
2. When $\mu = 4\hat{\mu}$, i.e. when the guess is slightly too low, the algorithm $\text{ALG}_{\hat{\mu}}$ has unbounded competitiveness.

Distortion-oblivious algorithms. Based on the previous discussion on robust algorithms, we desire a stronger guarantee than simple robustness: we would like a *single* algorithm which works well for every input, with a competitive ratio that is tailored to the input's distortion μ (rather than a distortion cap $\hat{\mu}$). Formally, the guarantee provided by robustness is

$$\boxed{\forall \mu > 1 : \exists\, \text{ALG}_\mu : \text{ALG}_\mu \text{ is } O(f(\mu))\text{-competitive on inputs with distortion } \mu}$$

whereas the stronger guarantee would be

$$\boxed{\exists\, \text{ALG} : \forall \mu > 1 : \text{ALG is } O(f(\mu))\text{-competitive on inputs with distortion } \mu}$$

We call an algorithm with this stronger guarantee a **distortion-oblivious** algorithm. To conclude, the advantages of a distortion-oblivious algorithm ALG over the robust algorithms $\left\{\text{ALG}_{\hat{\mu}}\right\}_{\hat{\mu}>1}$ of [1] are:

1. When $\mu \ll \hat{\mu}$, ALG would be $O(f(\mu))$-competitive while $\text{ALG}_{\hat{\mu}}$ would be $\Omega(\hat{\mu})$-competitive.
2. When $\mu > \hat{\mu}$, ALG would remain $O(f(\mu))$-competitive while $\text{ALG}_{\hat{\mu}}$ would break down completely.

1.1 Our Results

In this paper, we present the first distortion-oblivious algorithms for scheduling with the goal of minimizing total flow time.

1. We present the deterministic, distortion-oblivious **ZigZag** algorithm. For every μ, **ZigZag** is $O(\mu \log \mu)$-competitive for all inputs of distortion μ (with both underestimations and overestimations).
2. We show an $\Omega(\mu)$ lower bound on competitiveness for every randomized algorithm for inputs with distortion of μ. This implies that the our two algorithms have a nearly-optimal competitive ratio.

A salient feature of the **ZigZag** algorithm is that it manages to completely ignore the distortion of jobs observed through processing. An alternative approach could be to design an algorithm which attempts to learn the distortion of the input from those observations; using this technique we obtain the following algorithm.

3. We present the deterministic, distortion-oblivious algorithm **DL**. For every μ, **DL** is $O(\mu \log^2 \mu)$-competitive for inputs of distortion μ in which processing times are only underestimated (and never overestimated).

Note that this algorithm is unable to handle overestimations, and is thus less general than **ZigZag**; this is an inherent result of trying to learn from observed distortions, which we discuss further in Subsection 1.2.

The **ZigZag** algorithm is superior to **DL** in both competitive ratio and generality (as **ZigZag** supports both underestimations and overestimations); the paper thus focuses on presenting and analyzing **ZigZag**. We still present and analyze **DL** in the appendix, as we believe that the relevant techniques could be of independent interest.

The nearly-linear competitive ratios of our algorithms improves upon the best previously-known result in [1], which requires knowing μ in advance (non-oblivious) and even then is $O(\mu^2)$-competitive. As our lower bound shows, our algorithms are optimal up to logarithmic factors.

1.2 Our Techniques Both **DL** and **ZigZag** rely on the two scheduling strategies **SEPT** and **SR**. On their own, each of these strategies has an unbounded competitive ratio even in the presence of constant distortion; however, each strategy has a different merit which **DL** and **ZigZag** can exploit.

The first strategy is **SEPT** – scheduling according to Shortest Estimated Processing Time. In this strategy, we divide jobs into exponential classes according to estimated processing time, and always work on a job from the smallest class (i.e. shortest estimate).

While this strategy has unbounded competitive ratio even without the presence of distortion (see Appendix C), it has the merit of always working on a job of the lowest class – that is, the most cost-effective job available.

The second strategy is **SR** ("special rule"), presented in [8]. The strategy distinguishes between jobs that have not been processed at all ('full') and jobs that have ('partial'). This strategy always works on the partial job of the minimal class, unless there are at least two full jobs of lower class: in that case, the one with the lowest class is marked as partial.

The main merit of this strategy is that it maintains that the number of partial jobs is at most a constant time the number of full jobs. This ensures that the algorithm never reaches a bad state in which almost all jobs are partial and nearly-complete; in such a state, the optimal solution could finish those nearly-complete jobs and be well ahead of the algorithm. However, as previously stated, the **SR** strategy fails to be robust: Appendix C shows that its competitive ratio is unbounded even in the presence of constant distortion ($\mu = 4$).

The DL algorithm. Consider the case in which a full job r is of a smaller class than the minimum-class partial job q. If the algorithm requires a second full job of class smaller than q in order to switch to r, it follows the **SR** strategy. If the algorithm allows the second full job to be of *any* class, this is very similar to simply following the **SEPT** strategy. It turns out that a middle ground that leads to robustness is to require a full job which is up to $\log \mu$ classes above q. The **DL** algorithm attempts to implement this rule, but it does not know μ; instead, **DL** uses the maximum distortion seen thus far.

This turns out to be sufficient when there are only underestimations (which is when **DL** is $O(\mu \log^2 \mu)$-competitive), but fails in the case of overestimations. An intuitive explanation for this is that when the adversary is given the ability to underestimate and overestimate jobs, the underestimations will occur in the jobs chosen by the *algorithm* (and thus it will learn their distortion well), while the overestimations will occur in the jobs chosen by the *optimal solution* (and thus the algorithm will not learn of their overestimation).

The ZigZag algorithm. This failure of **DL** in the case of overestimations seems inherent to any algorithm that attempts to learn the distortion from the input. The **ZigZag** algorithm thus takes a different approach, and ignores observed distortion completely.

Our **ZigZag** algorithm maintains a set of partial jobs, i.e. jobs that have been processed to some degree. Each such job is either a ZIG job or a ZAG job. The algorithm always works on the partial job of the lowest class; however, under some circumstances this partial job could decide to "appoint" a full job of a lower class to be partial. ZIG jobs always appoint ZAG jobs, and do so according to **SEPT**. ZAG jobs always appoint ZIG jobs, and do so according to **SR**.

However, for the algorithm to be competitive, we require another component – the ZIGZAG jobs. When a full job is released between a ZAG job and the ZIG job that appointed it, the ZAG job turns into a ZIGZAG job. ZIGZAG jobs still appoint ZIG jobs (like they did as ZAG jobs), but do so according to **SEPT** rather than **SR**.

1.3 Related Work The observation that exact processing times are usually not available in practice also motivated the *nonclairvoyant model*. In this model, the algorithm is not given the processing times of jobs, and must process them blindly; the algorithm only becomes aware of the processing time of a job when the job is completed. In a sense, the nonclairvoyant model goes to the other extreme – it assumes that *nothing* is known about the processing time of a job. This strictness of the model comes at a cost – the best algorithm for nonclairvoyant flow-time scheduling in [7] requires randomization, and has a competitive ratio that grows with the number of jobs n (specifically, $O(\log n)$). In [20], a matching lower bound of $\Omega(\log n)$-competitiveness was shown for randomized algorithms, as well as a lower bound of $\Omega(n^{1/3})$-competitiveness for deterministic algorithms. Other considerations of the nonclairvoyant scheduling model appear in [14, 7, 15, 6, 12, 11].

A similar model to the model considered in this paper is the semiclairvoyant model in [8]. In this model, one knows the power-of-2 class of the processing time, but not the exact processing time. This model is different from

the robust model since the distortion is constant and conforms to the boundaries of classes.

A generalization of the total flow time goal is total *weighted* flow time, in which each job also has a weight which scales its flow time. This goal function was studied in e.g. [10, 5, 2, 9]. The best-known upper bounds are logarithmic in the parameters of the input (processing-time ratio, weight ratio, density ratio). Bansal and Chan [4] showed that dependence on these parameters is necessary.

The robust model for scheduling was introduced in [1]. For minimizing total flow time, the paper introduced μ-robust, $O(\mu^2)$ competitive algorithms for every μ. In addition, the paper also introduced robust algorithms for weighted flow time with polynomial dependence (quadratic and cubic) on the distortion and logarithmic dependence on the parameters. All of these robust algorithm require a priori knowledge of the distortion.

A related emerging field is algorithms with predictions, in which algorithms are augmented with predictions of varying accuracy which relate to the incoming input. Such algorithms are expected to produce a competitive ratio which is a function of the accuracy of the predictions. In particular, scheduling with predictions for minimizing total completion time was considered in [21, 13] (these papers assume that all jobs are given at time 0). Scheduling with predictions has also been studied in the speed-scaling model [3], under stochastic arrival assumptions [18] and for load balancing [16, 17]. Additional work on algorithms with predictions can be found in [19].

Paper Organization. The **ZigZag** algorithm is presented and analyzed in Section 3. The **DL** algorithm is presented and analyzed in Appendix A. The lower bound for inputs with distortion μ is given in Section 4. Bad cases for various existing algorithms are shown in Appendix C.

2 Preliminaries

In the problem we consider, jobs arrive over time. Each job q must be processed for p_q time units until its completion (p_q is called the *processing time* or *volume* of q). We denote the release time of q by r_q.

Upon the release of a job q, the algorithm is *not* given the processing time p_q; instead, the algorithm is given an estimate $\tilde{p}_q$ to the processing time of q.

We define $\mu_1 := \max_{\text{job } q} \frac{p_q}{\tilde{p}_q}$, the maximum underestimation factor of a job in the input. We also define $\mu_2 := \max_{\text{job } q} \frac{\tilde{p}_q}{p_q}$, the maximum overestimation factor of a job in the input. (we demand that $\mu_1, \mu_2 \geq 1$; if this is not the case for one of these factors, define that factor to be 1.) It thus holds for every job q that $\frac{\tilde{p}_q}{\mu_2} \leq p_q \leq \mu_1 \cdot \tilde{p}_q$. Finally, we define $\mu := \mu_1 \cdot \mu_2$, the distortion parameter of the input. Note that these parameters μ_1, μ_2, μ are functions of the online input, and thus the algorithm has no prior knowledge of them.

The goal of an algorithm ALG is to minimize the *total flow time*, which is the sum over jobs of their time in the system, or $\sum_{\text{job } q}(C_q^{\text{ALG}} - r_q)$ where C_q^{ALG} is the completion time of q in the algorithm. Denoting by $\delta(t)$ the number of pending jobs in the algorithm at t (i.e. jobs that were released but not yet completed), an equivalent definition of flow time is $\int_0^\infty \delta(t)dt$.

The following definition of job classes is used throughout the paper.

DEFINITION 2.1. (JOB CLASS) *We define the* class *of a job q, denoted ℓ_q, to be the unique integer i such that* $\tilde{p}_q \in [2^i, 2^{i+1})$.

Note that this definition refers to the *estimated* processing times provided in the input (rather than actual processing times, or remaining processing times). In particular, the class of a job does not change over time.

3 The ZigZag Algorithm

In this section, we describe and analyze the **ZigZag** algorithm, a distortion-oblivious algorithm which is $O(\mu \log \mu)$-competitive for every input with distortion at most μ, for every $\mu > 1$.

3.1 The Algorithm We now describe the following algorithm for the robust scheduling problem. The algorithm marks some set of the pending jobs as *partial* jobs. These partial jobs are the only jobs that may undergo processing in the algorithm; the remaining jobs, called *full* jobs, have not undergone any processing.

The algorithm always works on the minimum-class partial job q (there can be at most one partial job in any class). If there exist full jobs of lower classes than q, the job q might choose to "appoint" the minimum-class

full job to be partial (this newly-appointed job is now the minimum-class partial job, and will thus be processed next).

This decision depends on the type of q as a partial job. Each partial job is one of the three types ZIG, ZAG and ZIGZAG.

1. If q is a ZIG job, it immediately appoints any smaller-class job, and marks this job as ZAG.
2. If q is a ZAG job, it only appoints the minimum-class full job when there exist (at least) two jobs of class less than q. This appointed job is then marked as ZIG.
3. If q is a ZIGZAG job, it immediately appoints any smaller-class job (like ZIG jobs do), but marks this job as ZIG (like ZAG jobs do).

ZIG and ZAG jobs are such immediately from the point of their appointment. ZIGZAG jobs are created in the following way: when a ZAG job q is the minimal-class partial job, and there exists a full job between the class of q and the next-higher-class partial job, the ZAG job q morphs into a ZIGZAG job.

ALGORITHM 3.1. **ZigZag** Algorithm

while there exist pending jobs **do**
  if there is no partial job in the algorithm **then**
    Mark an arbitrary minimum-class pending job as a partial ZIG job.
    continue to the next iteration of the loop.

  Let q be the minimum-class partial job in the algorithm.
  Let q' be the minimum-class job in the algorithm (partial or full).

  if q is a ZIG job **then**
    if $\ell_{q'} < \ell_q$ **then**
      Mark q' as a partial ZAG job.
      continue to the next iteration of the loop.

  else if q is a ZAG job **then**
    Let $\hat{q}$ be the partial job of the smallest class such that $\ell_{\hat{q}} > \ell_q$.
    if there exists a full job of class in $[\ell_q, \ell_{\hat{q}}]$ **then**
      Change q from a ZAG job to a ZIGZAG job.
      continue to the next iteration of the loop.
    if $\ell_{q'} < \ell_q$ **and** there exists some other job $q'' \neq q'$ such that $\ell_{q''} < \ell_q$ **then**
      Mark q' as a partial ZIG job.
      continue to the next iteration of the loop.

  else ▷ *q is a* ZIGZAG *job*
    if $\ell_{q'} < \ell_q$ **then**
      Mark q' as a partial ZIG job.
      continue to the next iteration of the loop.
    Process q.

A visualization of the **ZigZag** algorithm can be found in Figure 1. In this visualization, the pending jobs are shown inside their classes (the jobs appear as geometric shapes). The jobs can be either full jobs (hollow circles), ZIG jobs (blue squares), ZAG jobs (red rhombus), or ZIGZAG jobs (purple star). The gray arrows show appointment, where each partial job has necessarily been appointed by the next-higher-class partial job. Note that there is always at most one partial job per class, and note the alternation between ZIG (blue) jobs and ZAG or ZIGZAG jobs (red and purple). In addition, note that there is always a full job between a ZIG job and a ZIGZAG job appointed by it. Similarly, there is always a full job between a ZAG job and a ZIG job appointed by it.

An additional visualization of the evolution of job types is given in Figure 2.

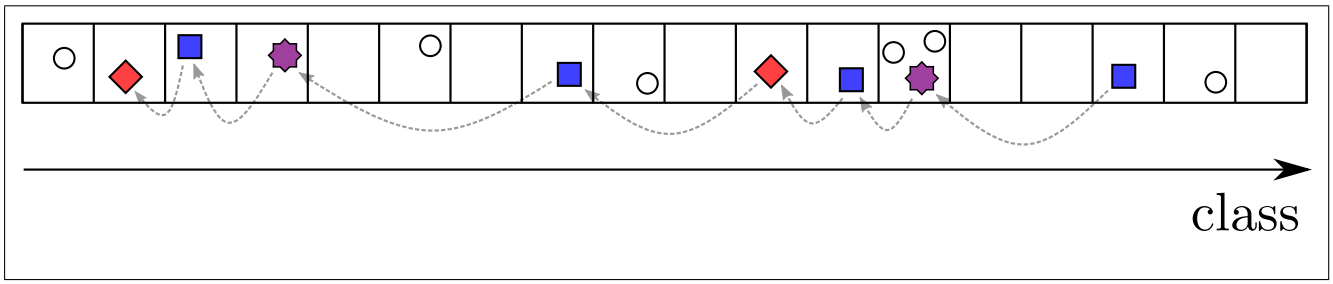
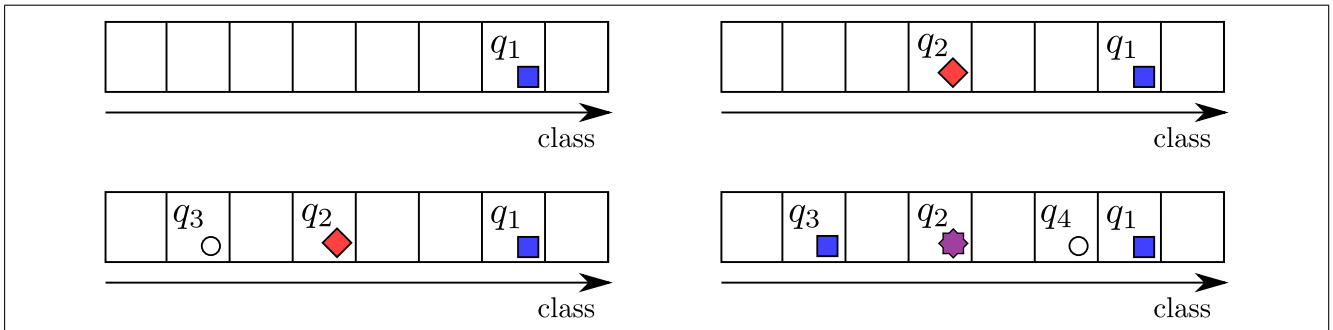


Figure 1: A Possible State of the **ZigZag** Algorithm (square=ZIG, rhombus=ZAG, star=ZIGZAG, circle=full)



In the images above, from left to right:

- q_1 is released and is immediately appointed ZIG.
- q_2 is released, and is immediately appointed ZAG by q_1.
- q_3 is released. q_2 does not appoint q_3, since there is only a single job of class less than ℓ_{q_2}.
- q_4 is released, which immediately turns q_2 from ZAG to ZIGZAG. Then, q_2 immediately appoints q_3 as ZIG.

Figure 2: A Possible Evolution of Jobs in Algorithm 3.1

In the remainder of this section, we prove the following theorem regarding Algorithm 3.1.

THEOREM 3.1. *For every μ, Algorithm 3.1 is $O(\mu \log \mu)$-competitive for inputs with distortion μ.*

3.2 Analysis We prove Theorem 3.1 through the following lemma, which states a property known as local competitiveness.

LEMMA 3.1. *At any time t, it holds that $\delta(t) \leq O(\mu \log \mu) \cdot \delta^*(t)$.*

If Lemma 3.1 holds, Theorem 3.1 follows simply through integrating over t. The remainder of this section focuses on proving Lemma 3.1; we henceforth fix time t for the remainder of this analysis. In addition, note that if $\delta^*(t) = 0$ then since the algorithm is non-idling, it must also be that $\delta(t) = 0$, and Lemma 3.1 holds. Thus, we henceforth assume that $\delta^*(t) \geq 1$.

3.2.1 Bounding $\delta(t)$ by Far-Behind Classes In this subsection, we bound the number of living jobs in the algorithm by the number of "far-behind" classes, which we define soon.

DEFINITION 3.1. (VOLUME AND LIVING JOBS NOTATION) *For every class i and time τ, we define:*

- *$V_{=i}(\tau)$ ($V_{\leq i}(\tau)$) to be the total remaining volume at τ of jobs of class exactly i (at most i) in the algorithm.*
- *Similarly, we define $V^*_{=i}(\tau)$ ($V^*_{\leq i}(\tau)$) to be the total remaining volume at τ of jobs of class exactly i (at most i) in the optimal solution.*
- *Finally, we define $\Delta V_{=i}(\tau) := V_{=i}(\tau) - V^*_{=i}(\tau)$ (and similarly $\Delta V_{\leq i}(\tau) := V_{\leq i}(\tau) - V^*_{\leq i}(\tau)$).*

We use similar subscript notation with δ to refer to the number of pending jobs of some class (or class range), and use superscript $*$ to refer to that amount in the optimal solution.

DEFINITION 3.2. (FAR-BEHIND CLASSES) *For every class i, we say that i is* far behind *at t if $\Delta V_{\leq i}(t) \geq \frac{2^i}{\mu_2}$. We also denote by S the set of far-behind classes at t.*

In this subsection, we prove the following lemma.

LEMMA 3.2. $\delta(t) \leq O(\mu) \cdot \delta^*(t) + O(\mu) \cdot |S|$

We first make some observations regarding the algorithm.

OBSERVATION 3.1. *At every time τ, and for every class j, there exists at most one partial job of class j at τ.*

OBSERVATION 3.2. *At any time τ, denote by $q_1, q_2, \cdots, q_k$ the partial jobs in the algorithm, by order of decreasing class (q_1 has the largest class). Then each job in $\{q_1, q_3, q_5, \cdots\}$ is a ZIG job, and each job in $\{q_2, q_4, q_6, \cdots\}$ is either a ZAG job or a ZIGZAG job.*

OBSERVATION 3.3. *Since a job q becomes partial and until its completion, the algorithm only processes job q or jobs of class less than ℓ_q. Moreover, during that time interval no full job of class at least ℓ_q becomes partial.*

OBSERVATION 3.4. *Suppose a partial job q is being processed at time τ. Then there is at most one job of class less than ℓ_q (which, if exists, is necessarily full).*

DEFINITION 3.3. *For every time τ, we denote:*

- *the number of full jobs at t by $\delta^f(\tau)$.*
- *the number of partial jobs at t by $\delta^p(\tau)$.*

PROPOSITION 3.1. *Consider any four partial jobs q_1, q_2, q_3, q_4 in the algorithm at t such that:*

- *$\ell_{q_1} < \ell_{q_2} < \ell_{q_3} < \ell_{q_4}$*
- *q_1, q_2, q_3, q_4 are consecutive: that is, for every i there exists no partial job q' such that $\ell_{q_i} < \ell_{q'} < \ell_{q_{i+1}}$.*

Then there exists a full job q such that $\ell_{q_1} \leq \ell_q \leq \ell_{q_4}$.

Proof. Using Observation 3.2, either q_1, q_3 are ZIG jobs or q_2, q_4 are ZIG jobs.
Assume henceforth that q_1, q_3 are ZIG jobs. This implies that q_2 is either a ZAG job or a ZIGZAG job.
Note that among q_1, q_2, q_3, q_3 became partial first, then q_2 and then q_1; any other order would contradict Observation 3.3.
If q_2 is a ZAG job, then consider the point in time t' in which q_1 became partial. At that point, there was no pending job of class lower than ℓ_{q_1}. Moreover, there was no partial job of class lower than ℓ_{q_2} at t' (otherwise, that job would be alive in t, contradicting the fact that q_1, q_2 are consecutive). This implies that q_1 became partial when q_2 was the minimum-class partial job, as part of Line 17 in the algorithm. Line 17 ran at t' because there had been at least two full jobs of class smaller than ℓ_{q_2}. Thus, after q_1 became partial, there remained a full job q in the class range $[\ell_{q_1}, \ell_{q_2})$. Applying Observation 3.3 to q_1, we have that q is still full at t.
Otherwise, q_2 is a ZIGZAG job. In this case, let t' be the point in time in which q_2 became a ZIGZAG job. This happened since there was a full job q in the class range $[\ell_{q_2}, \ell_{q_3}]$; applying Observation 3.3 to q_2 implies that q is still full at t.
Overall, we have that if q_1, q_3 are ZIG jobs, then there exists a full job in the class range $[\ell_{q_1}, \ell_{q_3}]$. In the second case, in which q_2, q_4 are ZIG jobs, the same argument yields that there exists a full job in the class range $[\ell_{q_2}, \ell_{q_4}]$. Thus, in both cases the proposition holds.

The following corollary is immediate from Proposition 3.1 by partitioning the partial jobs into groups of four.

COROLLARY 3.1. *It holds that $\delta^p(t) \leq 4\delta^f(t) + 3$.*

Corollary 3.1 allows us to focus on bounding the total number of full jobs, which we now proceed to do.

LEMMA 3.3. *It holds that*

$$\delta^f(t) \le O(\mu) \cdot \delta^*(t) + O(\mu) \cdot |S|$$

Proof. We omit t from V-notation and δ-notation in the following. Let $i_{\min}, i_{\max}$ be the minimum and maximum class of a job in the input, respectively.

$$\begin{aligned} \delta^f &= \sum_{i=i_{\min}}^{i_{\max}} \delta^f_{=i} \\ &\le \sum_{i=i_{\min}}^{i_{\max}} \left\lfloor \frac{V_{=i}}{\frac{2^i}{\mu_2}} \right\rfloor \\ &\le \sum_{i=i_{\min}}^{i_{\max}} \left\lfloor \frac{\mu_2 V^*_{=i} + \mu_2 \cdot \Delta V_{=i}}{2^i} \right\rfloor \\ &\le \sum_{i=i_{\min}}^{i_{\max}} \left\lceil \frac{\mu_2 V^*_{=i}}{2^i} \right\rceil + \sum_{i=i_{\min}}^{i_{\max}} \left\lfloor \frac{\mu_2 \cdot \Delta V_{=i}}{2^i} \right\rfloor \end{aligned} \tag{3.1}$$

where the first inequality is due to the fact that the minimum volume of a full job of class i is $\frac{2^i}{\mu_2}$ and the third inequality is due to the simple arithmetic trait that $\lfloor x + y \rfloor \le \lceil x \rceil + \lfloor y \rfloor$.
Observe the expression $\sum_{i=i_{\min}}^{i_{\max}} \left\lceil \frac{\mu_2 V^*_{=i}}{2^i} \right\rceil$. If $\delta^*_{=i} \ge 1$, we have

$$\left\lceil \frac{\mu_2 V^*_{=i}}{2^i} \right\rceil \le \frac{\mu_2 V^*_{=i}}{2^i} + 1 \le \frac{\mu_2}{2^i} \cdot 2^{i+1} \mu_1 \delta^*_{=i} + 1 = 2\mu\delta^*_{=i} + 1 \le (2\mu + 1)\delta^*_{=i}$$

Otherwise, $\delta^*_{=i} = 0$, in which case $\left\lceil \frac{\mu_2 V^*_{=i}}{2^i} \right\rceil = 0 = (2\mu + 1)\delta^*_{=i}$.
Thus, $\sum_{i=i_{\min}}^{i_{\max}} \left\lceil \frac{\mu_2 V^*_{=i}}{2^i} \right\rceil \le (2\mu + 1)\delta^*$. Plugging into Equation (3.1), we have

$$\begin{aligned} \delta^f &\le (2\mu + 1)\delta^* + \sum_{i=i_{\min}}^{i_{\max}} \left\lfloor \frac{\mu_2 \cdot \Delta V_{\le i} - \mu_2 \cdot \Delta V_{\le i-1}}{2^i} \right\rfloor \\ &\le (2\mu + 1)\delta^* + \sum_{i=i_{\min}}^{i_{\max}} \left(\left\lfloor \frac{\mu_2 \Delta V_{\le i}}{2^i} \right\rfloor - \left\lfloor \frac{\mu_2 \Delta V_{\le i-1}}{2^i} \right\rfloor \right) \\ &= (2\mu + 1)\delta^* + \sum_{i=i_{\min}}^{i_{\max}-1} \left(\left\lfloor \frac{\mu_2 \Delta V_{\le i}}{2^i} \right\rfloor - \left\lfloor \frac{\mu_2 \Delta V_{\le i}}{2^{i+1}} \right\rfloor \right) \end{aligned} \tag{3.2}$$

where the second inequality is due to the arithmetic trait that $\lfloor x - y \rfloor \le \lfloor x \rfloor - \lfloor y \rfloor$, and the equality is through rearranging a telescopic sum, while observing that $\Delta V_{\le i_{\min}-1} = \Delta V_{\le i_{\max}} = 0$ (since the algorithm is non-idling).
Now, we claim that for every i it holds that $\Delta V_{\le i} \le 2^{i+1}\mu_1$. To see this, consider the last time before t in which we worked on a job of class strictly more than i, and denote this time by t_i. At t_i, there was at most one pending job of class at most i (Observation 3.4), and that job thus had at most $2^{i+1}\mu_1$ volume. Since from t_i the algorithm only worked on jobs of class at most i, $\Delta V_{\le i}(t) \le \Delta V_{\le i}(t_i) \le V_{\le i}(t_i) \le 2^{i+1}\mu_1$.
Thus, observe that the expression $\left\lfloor \frac{\mu_2 \Delta V_{\le i}}{2^i} \right\rfloor - \left\lfloor \frac{\mu_2 \Delta V_{\le i}}{2^{i+1}} \right\rfloor$ is:

- maximized when $\Delta V_{\le i} = 2^{i+1}\mu_1$, and its value then is at most $\mu + 1$.
- only positive when $\Delta V_{\le i} \ge \frac{2^i}{\mu_2}$, i.e. when i is far behind.

From these two observations, we can bound $\left\lfloor \frac{\mu_2 \Delta V_{\leq i}}{2^i} \right\rfloor - \left\lfloor \frac{\mu_2 \Delta V_{\leq i}}{2^{i+1}} \right\rfloor$ by $(\mu+1) \cdot \mathbb{I}(i \text{ is far behind})$. Plugging into Equation (3.2), we get that the weight of the full jobs δ^f is at most

$$O(\mu) \cdot \delta^* + O(\mu) \cdot |S|$$

as required.

We can now prove Lemma 3.2.

Proof of Lemma 3.2. The lemma results immediately from Corollary 3.1 and Lemma 3.3.

3.2.2 Bounding Far-Behind Classes To complete the bounding of the algorithm's cost, it is thus enough to bound the size of the set S, i.e. number of far-behind classes.

LEMMA 3.4. $|S| \leq O(\log \mu) \cdot \delta^*(t)$.

DEFINITION 3.4. *Define* $\sigma := \lceil \log \mu \rceil + 1$.

Intuitively, σ is the minimum number such that a job q of class i necessarily has less processing time than a job q' of class $i+\sigma$:

$$p_{q'} \geq \frac{2^{i+\sigma}}{\mu_2} \geq \frac{2^{i+\log \mu+1}}{\mu_2} = \mu \cdot \frac{2^{i+1}}{\mu_2} = \mu_1 \cdot 2^{i+1} > p_q$$

We now perform a sparsification of S to obtain the set S' in the following manner.

1. Add the minimum class in S to S'.
2. While possible, add the minimum class that is greater than the last-added class by at least 2σ.

OBSERVATION 3.5. *Observe that:*

1. $|S| \leq 2\sigma \cdot |S'|$.
2. $|i_1 - i_2| \geq 2\sigma$ *for every* $i_1, i_2 \in S'$.

A visualization of this sparsification process is given in Figure 3. The first image of the figure shows the far-behind classes of S in red. The second image of the figure shows the classes of the sparsified set S' in red, where the classes of $S \backslash S'$ are faded (in this example $\sigma = 2$). The purple lines show the range in which classes are excluded. Note that the distance between any two classes in S' is at least 2σ (which equals 4 in this figure).

We henceforth focus on bounding $|S'|$.

DEFINITION 3.5. *For every class* i*, we denote by* t_i *the last time until* t *in which the algorithm worked on a job of class strictly more than* i *(*$t_i := 0$ *if this never happened).*

PROPOSITION 3.2. *If class* i *is far behind at* t*, then there exists a pending job at* t *of some class in* $(i-\sigma, i]$.

Proof. Consider time t_i, the last time before t that the algorithm worked on a job of class strictly more than i. Note that during $(t_i, t]$ the algorithm only worked on jobs of class at most i, and i is far behind at t; thus, the total volume of jobs of class at most i at t_i must be at least $\frac{2^i}{\mu_2}$.
Now note that at t_i the algorithm worked on the minimum-class partial job q which is of class more than i. If this job q is ZIG or ZIGZAG at t_i, this is a contradiction, since it would imply that there are no jobs of class at most i at t_i. Thus, the job q is a ZAG job – but this implies that there is only a single job q' of class at most i at t_i. This job must thus have at least $\frac{2^i}{\mu_2}$ volume, which would imply that its class is greater than $i-\sigma$, and thus in $(i-\sigma, i]$.
Now, suppose for contradiction that no job of class in $(i-\sigma, i]$ exists at t. This implies that at some time τ in $(t_i, t]$, the last job of this class range was completed. At that time, there was at most one other job of class $\leq i$,

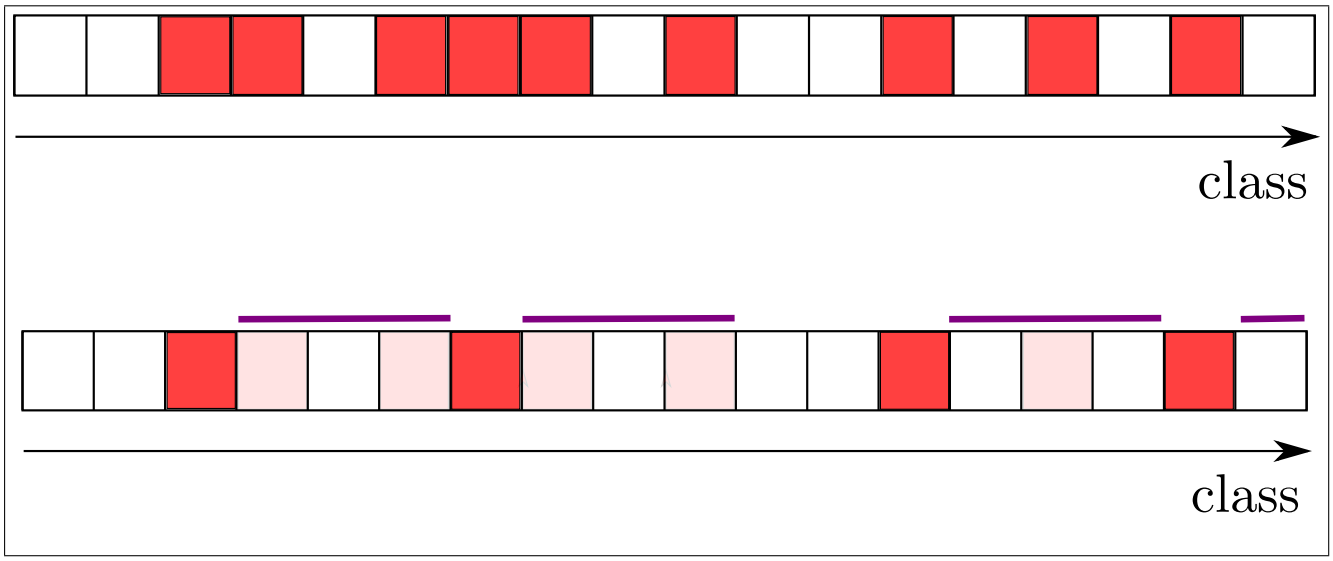


Figure 3: The Far-Behind Sparsification Process

and that job (if it exists) was of class $\leq i-\sigma$, and thus had strictly less than $\frac{2^i}{\mu_2}$ volume. Since $\tau > t_i$, we have

$$\Delta V_{\leq i}(t) \leq \Delta V_{\leq i}(\tau) < \frac{2^i}{\mu_2}$$

which is a contradiction to i being far behind at t.

PROPOSITION 3.3. *If a* ZIG *job q of some class i is pending at t, it holds that $\Delta V_{\leq i-1}(t) \leq 0$.*

Proof. Observation 3.3 implies that from the time q became partial, the algorithm only worked on q or on jobs of class at most $i-1$. Consider the last time t' in which the algorithm worked on q: at that time, there were no jobs of classes at most $i-1$, and from that time the algorithm only worked on jobs of class at most $i-1$. Thus,

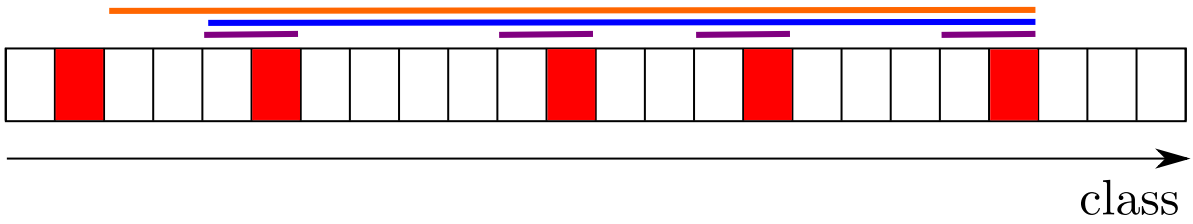
$$\Delta V_{\leq i-1}(t) \leq \Delta V_{\leq i-1}(t') \leq 0$$

PROPOSITION 3.4. *Let i, i' be two classes such that $i' \leq i-\sigma$, i' is far-behind, and there exists a full job of class i in the algorithm at t. Then the optimal solution has a job alive in the class range $(i', i]$.*

Proof. Denote by q the full job of class i in the algorithm at t. If q is pending in the optimal solution at t, we are done; henceforth assume that the optimal solution has completed q by time t.
We now aim to find a class $j \in [i-\sigma, i]$ such that $\Delta V_{\leq j}(t) \leq 0$, and claim that this would complete the proof. To prove this claim, assume that there exists such a j. Observe that $\Delta V_{\leq i'}(t) > 0$ (since i' is far behind); thus, it cannot be that $i' = j = i-\sigma$. Therefore, it holds that $i' < j$, which implies $\Delta V_{\in(i',j]}(t) = \Delta V_{\leq j}(t) - \Delta V_{\leq i'}(t) < 0$, where the subscript $\in (i', j]$ restricts the volume to jobs of classes in $(i', j]$. But this implies that the optimal solution must have a pending job at t of some class in $(i', j]$, and thus in $(i', i]$. Hence, the claim holds.
We now continue in proving the proposition. We consider $t_{i-\sigma}$ relative to r_q, and observe the following two cases.
Case 1: $r_q \geq t_{i-\sigma}$**.** In this case, note that at $t_{i-\sigma}$ there existed at most one job of class at most $i-\sigma$ due to Observation 3.4. Such a job had volume less than $2^{i-\sigma+1}\mu_1$ which is at most $\frac{2^i}{\mu_2}$. Thus, $V_{\leq i-\sigma}(t_{i-\sigma}) \leq \frac{2^i}{\mu_2}$.
In the time interval $(t_{i-\sigma}, t]$, the algorithm only worked on jobs of class at most $i-\sigma$, while the optimal solution started and completed q. This implies that

$$\Delta V_{\leq i-\sigma}(t) \leq \Delta V_{\leq i-\sigma}(t_{i-\sigma}) - p_q \leq 0$$

which completes the proof for this case.



This figure shows the proof of Lemma 3.5. The figure shows five far-behind classes in S'. Proposition 3.2 implies that there exists a pending job in each of the four purple-colored class segments. Proposition 3.1 implies that there is a full job in the blue-colored segment. Finally, Proposition 3.4 implies that the optimal solution has a pending job in the orange segment.

Figure 4: Proof of Lemma 3.5

Case 2: $r_q < t_{i-\sigma}$. In this case, consider the job q' being processed at $t_{i-\sigma}$. From the definition of $t_{i-\sigma}$, it must be that $\ell_{q'} > i - \sigma$. If there exists no job of class $\leq i - \sigma$ at $t_{i-\sigma}$, we are done, since the algorithm only works on such jobs in $(t_{i-\sigma}, t]$, and thus

$$\Delta V_{\leq i-\sigma}(t) \leq \Delta V_{\leq i-\sigma}(t_{i-\sigma}) \leq 0$$

Otherwise, there exists a job r of class at most $i - \sigma$ at $t_{i-\sigma}$, which implies that q' is a ZAG job at $t_{i-\sigma}$. Let q'' be the consecutive partial job to q', and note that q'' is a ZIG job (observe that a ZAG job always has a consecutive job). Now note that:

- $\ell_{q'} < i$; otherwise, the existence of both r and q would prevent q' from being processed at $t_{i-\sigma}$.
- $\ell_{q''} < i$; otherwise, q' (which is of class less than i) would have become a ZIGZAG job by seeing q, in contradiction to being ZAG at $t_{i-\sigma}$.

Since the algorithm does not work on a job of class more than $i - \sigma$ after $t_{i-\sigma}$, the ZIG job q'' remains pending at t. Using Proposition 3.3 implies that $\Delta V_{\leq \ell_{q''}-1}(t) \leq 0$. Since $\ell_{q''} - 1 \in [i - \sigma, i - 1]$, we are done.

LEMMA 3.5. *Let $i_1, i_2, \cdots, i_5 \in S'$ be five classes such that $i_1 < i_2 < \cdots < i_5$. Then there exists a pending job in the optimal solution at t of class in the range $[i_1, i_5]$.*

Proof. Applying Proposition 3.2 to $i_2, \cdots, i_5$ implies that there exist four jobs $q_2, \cdots, q_5$ such that for every $j \in \{2, 3, 4, 5\}$ it holds that $\ell_{q_j} \in (i_j - \sigma, i_j]$ (since the distance between any two of the five classes is at least 2σ, these four jobs are distinct).

If a job $q \in \{q_2, \cdots, q_5\}$ is a full job, we apply Proposition 3.4 to the far-behind class i_1 and to q to obtain that the optimal solution has a pending job in the class range $(i_1, \ell_q]$, which is contained in $[i_1, i_5]$, thus completing the proof.

Otherwise, assume that $\{q_2, \cdots, q_5\}$ are all partial jobs. Consider the four consecutive partial jobs starting with q_2, denoted as q_2, r_1, r_2, r_3, such that $\ell_{q_2} < \ell_{r_1} < \ell_{r_2} < \ell_{r_3}$. It necessarily holds that $\ell_{r_3} \leq \ell_{q_5}$. We apply Proposition 3.1 to obtain a full job q such that $\ell_q \in [\ell_{q_2}, \ell_{r_3}]$, which is contained in $(i_2 - \sigma, i_5]$. As before, we apply Proposition 3.4 to i_1 and q which yields that the optimal solution has a pending job in the class range $[i_1, i_5]$.

The following corollary is immediate from Lemma 3.5.

COROLLARY 3.2. $|S'| \leq 5\delta^*(t) + 4$

We now return to proving Lemma 3.4.

Proof of Lemma 3.4. Results immediately from Observation 3.5 and Corollary 3.2.

We can now complete the proof of Lemma 3.1, which implies Theorem 3.1.

Proof of Lemma 3.1. The lemma results immediately from Lemmas 3.2 and 3.4.

4 Lower Bound

In this section, we show a lower bound for the robust scheduling model. This lower bound shows that a sublinear dependence on the distortion is impossible in either robust or distortion-oblivious algorithms.

THEOREM 4.1. *For every choice of distortion parameter μ, every randomized (or deterministic) algorithm is $\Omega(\mu)$-competitive on inputs with distortion at most μ.*

The proof of Theorem 4.1 appears in Appendix B.

5 Discussion and Open Problems

In this paper, we presented the first distortion-oblivious algorithms for total flow time, which also have a nearly optimal competitive ratio. Thus, this paper essentially closes the problem of robustness/distortion-obliviousness for total flow time.

It would be interesting to see whether distortion-oblivious algorithms could be designed for other scheduling goals. A prominent example is *weighted* flow time: while [1] introduced robust algorithms for this problem, no distortion-oblivious algorithms are known. One could also consider other goals, such as minimizing mean stretch (ratio of flow time to processing time). Finally, extending the distortion model to multiple machines and obtaining distortion-oblivious algorithms seems like another natural direction.

References

[1] Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. *CoRR*, abs/2103.05604, 2021.

[2] 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.

[3] Etienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling, 2020.

[4] 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.

[5] Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. *ACM Trans. Algorithms*, 3(4):39, 2007. also in SODA 2003: 508-516.

[6] Nikhil Bansal, Kedar Dhamdhere, Jochen Könemann, and Amitabh Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. *Algorithmica*, 40(4):305–318, 2004.

[7] 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.

[8] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Semi-clairvoyant scheduling. *Theor. Comput. Sci.*, 324(2-3):325–335, 2004.

[9] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *Journal of Discrete Algorithms*, 4(3):339 – 352, 2006. Special issue in honour of Giorgio Ausiello.

[10] 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.

[11] 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.

[12] 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.

[13] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, page 285–294, New York, NY, USA, 2021. Association for Computing Machinery.

[14] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance [scheduling problems]. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 214–221, 1995.

[15] Jae-Hoon Kim and Kyung-Yong Chwa. Non-clairvoyant scheduling for weighted flow time. *Inf. Process. Lett.*, 87(1):31–37, 2003.

[16] 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.

[17] Thomas Lavastida, Benjamin Moseley, R. Ravi, and C. Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. *ArXiv*, abs/2011.11743, 2020.

[18] 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.

[19] Michael Mitzenmacher and Sergei Vassilvitskii. *Algorithms with Predictions*, page 646–662. Cambridge University Press, 2021.

[20] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47, 1994.

[21] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

[22] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

A The DL Algorithm

In this section, we present and analyze the **DL** algorithm, a distortion-oblivious algorithm which, for every μ, is $O(\mu \log^2 \mu)$-competitive for inputs with distortion μ which contain only underestimations.

A.1 The DL Algorithm

Description of DL. As in the **ZigZag** algorithm, the **DL** algorithm maintains a set of partial jobs, which are the only jobs which undergo processing (and there is at most one such partial job per class).

At any point in time, the algorithm processes the minimum-class partial job q, unless:

1. there exists a pending (full) job of a lower class, and
2. there exists a third pending job of class less than $\ell_q + \hat{\sigma}$.

If both conditions hold, the minimum-class full job is marked as partial. The global parameter $\hat{\sigma}$ is the algorithm's estimate for the parameter σ; this parameter $\sigma = \lceil \log \mu \rceil + 1$ is as defined in Definition 3.4 for **ZigZag**. The parameter $\hat{\sigma}$ is updated according to the distortion witnessed by the algorithm (through processing jobs for more than their estimated processing times).

The **DL** algorithm is given in Algorithm A.1.

ALGORITHM A.1. **DL** Algorithm

while there exist pending jobs **do**
  if there is no partial job **then**
    Mark the minimum-class pending job as partial.
    continue to the next iteration of the loop.
  Let q be the minimum-class partial job in the algorithm.
  if there exist two distinct full jobs q', q'', such that $\ell_{q'} < \ell_q$ and $\ell_{q''} < \ell_q + \hat{\sigma}$ **then**
    Mark the minimum-class full job as partial.
    continue to the next iteration of the loop.
  Process q.
  if q has been processed for more than $2^i \cdot \tilde{p}_q$ time for some i **then**
    Set $\hat{\sigma} \leftarrow \max(\hat{\sigma}, i+2)$

THEOREM A.1. *For every μ, Algorithm A.1 is $O(\mu \log^2 \mu)$-competitive for inputs of distortion μ with no overestimations (i.e. $\mu_2 = 1$).*

A.2 Analysis The analysis of **DL** follows the general structure of the analysis of **ZigZag**. However, some of the lemmas and propositions require different proofs.

As in the analysis of the **ZigZag**, the main lemma shows local competitiveness, and immediately implies Theorem A.1.

LEMMA A.1. *Consider an input with distortion at most μ which is without overestimations (i.e. $\mu_2 = 1$). At any time t, it holds that $\delta(t) \le O(\mu \log^2 \mu) \cdot \delta^*(t)$.*

We henceforth fix a time t towards proving Lemma A.1, and (as in the **ZigZag** analysis) assume $\delta^*(t) \ge 1$.

A.2.1 Bounding $\delta(t)$ by Far-Behind Classes We use the same notation defined in Definition 3.1. In addition, we again define far-behind classes; the following is a restatement of Definition 3.2 where we note that $\mu_2 = 1$.

DEFINITION A.1. (RESTATEMENT OF DEFINITION 3.2) *For every class i, we say that i is* far behind *at t if $\Delta V_{\leq i}(t) \geq 2^i$.*
We also denote by S the set of far-behind classes at t.

The following lemma bounds the number of living jobs in the algorithm by the number of far-behind classes at t.

LEMMA A.2. $\delta(t) \leq O(\mu \log \mu)\delta^*(t) + O(\mu \log \mu) \cdot |S|$

We now focus on proving Lemma A.2.

We define σ as in Definition 3.4. This parameter σ is what the variable $\hat{\sigma}$ attempts to learn as the algorithm progresses; note that the value of $\hat{\sigma}$ is always at most σ.

PROPOSITION A.1. *Let q_1, q_2 be two partial jobs of classes i_1, i_2 respectively, such that $i_1 < i_2$. Then there exists a full job in the range $[i_1, i_2 + \sigma]$.*

Proof. Assume that there is no other partial job in the range (i_1, i_2) (otherwise, replace q_2 with this job and continue with the proof). It must be that q_1 became partial after q_2. When q_1 became partial, q_2 was the minimum-class partial job. Since q_2 was not processed (and q_1 was processed instead) there must be another full job q_3 (other than q_2) at that time of class at most $i_2 + \hat{\sigma}$ for the value of $\hat{\sigma}$ at that time, which is at most $i_2 + \sigma$. In addition, since q_1 was the minimum-class job at the time, the class of q_3 was at least i_1. This completes the proof.

We use the notation defined in Definition 3.3 to refer to the number of full/partial jobs. Note that Observation 3.1 for **ZigZag** applies to **DL** as well; thus, the following corollary holds.

COROLLARY A.1. *It holds that $\delta^p(t) \leq (\sigma + 2)\delta^f(t)$.*

Note that Lemma 3.3 for the **ZigZag** algorithm is true independently of the algorithm, and thus applies also for **DL**.

LEMMA A.3. (RESTATEMENT OF LEMMA 3.3) *It holds that*

$$\delta^f(t) \leq O(\mu) \cdot \delta^*(t) + O(\mu) \cdot |S|$$

We can now prove Lemma A.2.

Proof of Lemma A.2. Results immediately from observing that $\sigma = O(\log \mu)$, and applying Corollary A.1 and Lemma A.3.

A.2.2 Bounding Far-Behind Classes

We would now like to bound the number of far-behind classes $|S|$.

LEMMA A.4. $|S| \leq O(\log \mu) \cdot \delta^*(t)$.

We perform the same sparsification process as in the **ZigZag** analysis to obtain S', and note that Observation 3.5 applies to S'. We also define t_i for every i, as in Definition 3.5.

Note that the proof of Proposition 3.2 remains true for **DL** as well.

PROPOSITION A.2. (RESTATEMENT OF PROPOSITION 3.2) *If class i is far behind at t, then there exists a pending job at t of some class in $(i - \sigma, i]$.*

We would now like to prove Proposition 3.4 for **DL**.

PROPOSITION A.3. (RESTATEMENT OF PROPOSITION 3.4) *Let i, i' be two classes such that $i' \leq i - \sigma$, i' is far-behind, and there exists a full job of class i in the algorithm at t. Then the optimal solution has a job alive in the class range $(i', i]$.*

Proof. First, we require the following claim.
Claim: if there exists a class $j \in (i', i]$ such that $\Delta V_{\leq j}(t) \leq 0$, then we are done. To prove the claim, note that

$\Delta V_{\leq i'}(t) \geq 2^{i'} > 0$ since i' is far behind, which implies that

$$\Delta V_{\leq j, > i'}(t) < 0$$

and thus the optimal solution must have a job in class range $(i', j] \subseteq (i', i]$ as required. This completes the proof of the claim.

Returning to the proof of the proposition, if the optimal solution has a job of class i, then we are done. Henceforth assume that it has no such job.

Let q be the full job of class i in the algorithm. Denote by t_j the last point in time prior to t in which a job of class strictly more than j was processed in the algorithm. We now split into cases according to the release time of q.

Case 1: $r_q < t_{i-1}$. In this case, at t_{i-1} we have that q has already been released, yet a job r of class $\geq i$ is being processed. Thus, either the special rule is not being applied, or the special rule *is* being applied in skipping over q. In either case, there is no job of class $\leq i-1$ at t. Since from t_{i-1} onwards the algorithm only works on jobs of class $\leq i-1$, it must be that $\Delta V_{\leq i-1}(t) \leq 0$. Since $i-1 \in (i', i]$, the claim above implies that the proposition holds.

Case 2: $r_q \geq t_{i-\sigma}$. In this case, at time $t_{i-\sigma}$ there exists at most a single job of class at most $i-\sigma$, the volume of which is at most $2^{i-\sigma} \cdot \mu \leq 2^i$. Thus, $V_{\leq i-\sigma}(t_{i-\sigma}) \leq 2^i$. During the interval $(t_{i-\sigma}, t]$, the algorithm only worked on jobs of class at most $i-\sigma$, while the optimal solution spent at least 2^i time on q.

Thus, $\Delta V_{\leq i-\sigma}(t) \leq 0$. If $i' = i-\sigma$, this is a contradiction to i' being far behind. Otherwise, $i-\sigma \in (i', i]$, and thus the above claim implies that the proposition holds.

Case 3: $r_q \in [t_j, t_{j-1})$ for some $j \in (i-\sigma, i-1]$. In this case, consider time t_j, in which a job r of class $> j$ was being processed. If this job was processed without the special rule, then there is no living job in the algorithm of class $\leq j$ at t_j, which implies that $\Delta V_{\leq j}(t) \leq 0$. The claim above would thus imply that the proposition holds.

Otherwise, the special rule was applied, skipping over a full job r'. If the job r' is of class more than j, we are again done for the same reason. Assume therefore that r' is of some class j' which is at most j.

If $p_{r'} \leq 2^i$, then observe that $V_{\leq j}(t_j) \leq 2^i \leq p_q$. Since the algorithm only works on jobs of class at most j from t_j onwards, while the optimal solution spends p_q time on completing job q, we have that $\Delta V_{\leq j}(t) \leq 0$. The above claim would thus imply that the proposition holds; henceforth assume that $p_{r'} > 2^i$.

If during the interval $(t_j, t]$ the algorithm spends at most 2^i time on job r', then observe that at t_j the job r' is the only job alive of class $\leq j$. Thus, the entire interval $(t_j, t]$ was spent on jobs of class $\leq j$ born after t_j, except for at most 2^i time units. During the same interval, the optimal solution manages to complete the entire job q. Thus, it holds that $\Delta V_{\leq j}^{(t_j, t]}(t) \leq 0$, where the time interval in the superscript restricts the considered volume to jobs released in that interval. Now, note that the fact that $p_{r'} \geq 2^i$ implies that $j' > i-\sigma \geq i'$. Thus, at time t_j there are no jobs of class $\leq i'$. This implies that all pending jobs of class $\leq i'$ at t were released after t_j, and thus $\Delta V_{\leq i'}^{(t_j, t]}(t) \geq \Delta V_{\leq i'}(t) \geq 2^{i'}$, since i' is far behind. Thus, $\Delta V_{\leq j, > i'}^{(t_j, t]}(t) < 0$ which implies that the optimal solution has a living job in some class in $(i', j] \subseteq (i', i]$. This would complete the proof of the proposition; assume henceforth that the algorithm worked on r' for strictly more than 2^i time.

At time $t_{j'-1}$, the algorithm works on a job r'' of class $\geq j'$. But at that point, r' has already been processed for more than 2^i time units, and job q of class i has been released; thus, r'' is not being processed due to the special rule. This implies that there is no job of class $\leq j'-1$ at $t_{j'-1}$, and thus $\Delta V_{\leq j'-1}(t) \leq 0$. Now, note that the fact that $p_{r'} \geq 2^i$ implies that $j > i-\sigma \geq i'$; the claim thus applies and completes the proof of the proposition.

We can now prove an analogue of Lemma 3.5 for the **DL** algorithm.

PROPOSITION A.4. (ANALOGUE OF LEMMA 3.5) *Let i_1, i_2, i_3, i_4 be four consecutive classes in S'. Then there exists a job in the optimal solution of class in the range (i_1, i_4).*

Proof. First, we claim that there exists a full job q alive in the algorithm such that $\ell_q \in [i_2 - \sigma, i_3 + \sigma]$ we apply Proposition A.2 to i_2, i_3 to imply that there exist two pending jobs q_1, q_2 in the algorithm at t, such that $\ell_{q_1} \in (i_2 - \sigma, i_2]$ and $\ell_{q_2} \in (i_3 - \sigma, i_3]$ (since these class intervals are disjoint, we have that q_1, q_2 are

distinct).
If either one of q_1, q_2 is full at t, we choose q to be that job. Otherwise, both q_1, q_2 are partial, and we thus apply Proposition A.1 which implies that the full job q exists such that $\ell_q \in [\ell_{q_1}, \ell_{q_2} + \sigma] \subseteq [i_2 - \sigma, i_3 + \sigma]$. Observe that $\ell_q \geq i_2 - \sigma \geq i_1 + \sigma$; we thus apply Proposition A.3 to the far-behind class i_1 and the full job q, and conclude that the optimal solution has a pending job in the class range $(i_1, \ell_q) \subseteq (i_1, i_3 + \sigma) \subseteq (i_1, i_4)$. This completes the proof.

COROLLARY A.2. $|S'| \leq 4\delta^*(t)$

Proof of Lemma A.4. Results immediately from Observation 3.5 and Corollary A.2

Proof of Lemma A.1. Results immediately from Lemmas A.2 and A.4.

B Lower Bound - Proof of Theorem 4.1

In this section, we prove Theorem 4.1; the proof takes some ideas from [20]. For ease of presentation, we first introduce a warm-up *deterministic* lower bound of $\Omega(\mu)$. We then show the complete proof of Theorem 4.1 for randomized algorithms.

B.1 Warm-up: Deterministic Lower Bound We now loosely describe a simple, deterministic lower bound of $\Omega(\mu)$-competitiveness, before turning to the (somewhat more complex) randomized lower bound.

The adversary releases n jobs (for some large n) at time 0, with estimated processing times of 1. The adversary then waits until time $t := \frac{n\mu}{2}$.

Denoting by x_q the amount of time spent by the algorithm on job q until t, the processing time of q is $\min(x_q+1, \mu)$. That is, the algorithm never completes a job by time t unless it spends μ time on that job. Since the processing times of all jobs (which have estimate 1) is in $[1, \mu]$, the distortion is indeed at most μ.

For a job to have less than 1 unit of time remaining, the algorithm must spend more than $\mu - 1$ units of time on that job. Thus, from the definition of t it holds that $\delta(t, 1) \geq n - \frac{t}{\mu - 1} \geq \frac{n}{4}$ (recall the definition of $\delta(t, 1)$ from Definition C.1).

Meanwhile, note that there exist at least $\frac{n}{4}$ jobs q such that $x_q \geq \frac{\mu}{4}$ (assuming that the algorithm is non-idling). Denote the set of such jobs by R. The optimal solution could pick a subset $R' \subseteq R$ such that $|R'| = \frac{4n}{\mu}$ (using $\mu \geq 16$), and spend the time interval $[0, t]$ as follows:

- When the algorithm works on a job not in R', work on that job as well.
- When the algorithm works on a job in R', spend this time working on all jobs simultaneously (round robin).

Note that the total time devoted by the algorithm to jobs in R' is at least $|R'| \cdot \frac{\mu}{4} = n$; thus, the optimal solution is able to process every job not in R' for at least one unit of time more than the algorithm (due to the round robin). But this is enough to finish all jobs except for the jobs of R'; thus, $\delta^*(t) \leq \frac{4n}{\mu}$.

We thus have that $\frac{\delta(t,1)}{\delta^*(t)} = \Omega(\mu)$, and thus applying the bombardment technique (as stated in Lemma C.1) completes the deterministic lower bound.

B.2 Randomized Lower Bound: Proof of Theorem 4.1 We continue to show the randomized lower bound. We henceforth fix any distortion parameter μ and prove Theorem 4.1 for this distortion parameter.

We prove Theorem 4.1 using Yao's principle: we describe a distribution on μ-distorted inputs such that any deterministic algorithm is $\Omega(\mu)$-competitive against this distribution. The following proposition reduces the design of such a distribution to designing a distribution in which any deterministic algorithm is bad at some specific time t.

PROPOSITION B.1. *If there exists a distribution $\mathcal{D}$ on μ-distorted inputs and a time t such that for every algorithm* ALG *it holds that* $\frac{\mathbb{E}_{\mathcal{D}}(\delta(t,1))}{\mathbb{E}_{\mathcal{D}}(\delta^*(t))} \geq c$ *for some c, then there exists a distribution $\hat{\mathcal{D}}$ over μ-distorted inputs such that* $\frac{\mathbb{E}_{\hat{\mathcal{D}}}(\text{ALG})}{\mathbb{E}_{\hat{\mathcal{D}}}\text{OPT}} \geq \Omega(c)$.

Proof. The inputs of $\hat{\mathcal{D}}$ would behave exactly like the inputs of $\mathcal{D}$ until time t (and would have the same probability). From time t, all inputs would start a "bombardment" sequence, i.e. would release a job q with $\tilde{p}_q = p_q = 1$ every time unit for an arbitrarily large number of time units. An argument identical to that of Lemma C.1 completes the proof.

It remains to find such a distribution $\mathcal{D}$.

First, we describe the distribution $\mathcal{D}'$, which has unbounded distortion.

The distribution $\mathcal{D}'$. The distribution is defined with respect to the number of jobs k. The inputs all consist of releasing k jobs at time 0, each with predicted processing time 1. The real processing times of the jobs are i.i.d. random variables which are picked from the geometric distribution with mean 2 (i.e. with $p = \frac{1}{2}$). The adversary then waits for $2(k - k^{3/4})$ time units – we henceforth define $t := 2(k - k^{3/4})$.

PROPOSITION B.2. *For the distribution $\mathcal{D}'$, it holds that $\frac{\mathbb{E}_{\mathcal{D}'}(\delta(t,1))}{\mathbb{E}_{\mathcal{D}'}(\delta^*(t))} = \Omega(\log k)$.*

Proof. First, let's bound the $\mathbb{E}_{\mathcal{D}'}(\delta(t,1))$. As the jobs have integer processing times, we can assume without loss of generality that the algorithm does not devote fractional time units to jobs. Consider any time unit from 0 to t. Regardless of the job chosen for processing at that time, the probability that that job will be completed in this time unit is at most $\frac{1}{2}$ (it could be that no job is processed, in which case the probability is 0). Thus, the expected number of jobs completed in $2(k - k^{3/4})$ time units is at most $k - k^{3/4}$. Since jobs have integer processing times, jobs that are not completed have at least one time unit of processing remaining, and thus $\mathbb{E}_{\mathcal{D}'}(\delta(t,1)) \geq k^{3/4}$.
We now continue to bound the cost of the optimal solution. We make the following observations:

1. Denote the total processing time of the k jobs by Y. Note that Y is the sum of k independent geometric variables with mean 2, and thus has mean $2k$ and variance $2k$. Applying Chebyshev's inequality, $\Pr(Y > 2k + k^{3/4}) \leq O(1/\sqrt{k})$.
2. Defining $b := \frac{\log k}{4}$, the probability that a specific job of the k jobs has processing time more than b is $2^{-b} = k^{-1/4}$. Denoting by B the number of jobs with processing time more than b, it holds that $\mathbb{E}(B) = k^{3/4}$. Moreover, the variance of B is $O(k)$; thus, applying Chebyshev's inequality implies that $\Pr(B < \frac{k^{3/4}}{2}) \leq O(1/\sqrt{k})$.

Thus, with probability $1 - O(1/\sqrt{k})$, it holds that $Y \leq k + k^{3/4}$ and $B \geq \frac{k^{3/4}}{2}$. Thus, by time t the optimal solution can finish all jobs except for at most $O(\frac{k^{3/4}}{b})$ jobs with volume at least b each. Thus, we can bound the expected number of jobs in the optimal solution at t by:

$$\mathbb{E}_{\mathcal{D}'}(\delta^*(t)) \leq O\left(\frac{k^{3/4}}{\log k}\right) + O\left(\frac{1}{\sqrt{k}}\right) \cdot k \leq O\left(\frac{k^{3/4}}{\log k}\right)$$

which completes the proof of the proposition.

We can now prove Theorem 4.1.

Proof of Theorem 4.1. We construct the distribution $\mathcal{D}$ from the distribution $\mathcal{D}'$ by choosing $k = \left\lfloor 2^{\frac{\mu}{2}} \right\rfloor$ and conditioning on the event L that the processing times of jobs never exceed μ. This new distribution $\mathcal{D}$ thus has a distortion that is bounded by μ. Now, observe that:

$$\begin{aligned}\frac{\mathbb{E}_{\mathcal{D}}(\delta(t,1))}{\mathbb{E}_{\mathcal{D}}(\delta^*(t))} &= \frac{\Pr(L)\mathbb{E}_{\mathcal{D}'}(\delta(t,1)|L)}{\Pr(L)\mathbb{E}_{\mathcal{D}'}(\delta^*(t)|L)} \geq \frac{\mathbb{E}_{\mathcal{D}'}(\delta(t,1)) - \Pr(\bar{L})\mathbb{E}_{\mathcal{D}'}(\delta(t,1)|\bar{L})}{\mathbb{E}_{\mathcal{D}'}(\delta^*(t))} \\ &\geq \Omega(\log k) - \frac{\Pr(\bar{L})\mathbb{E}_{\mathcal{D}'}(\delta(t,1)|\bar{L})}{\mathbb{E}_{\mathcal{D}'}(\delta^*(t))}\end{aligned} \tag{B.1}$$

Now note that using the union bound on the processing times of jobs, $\Pr(\bar{L}) \leq k \cdot 2^{-\mu} \leq 1/k$. In addition, denoting by Y the sum of processing times of the k jobs (as before), and noting that $\mathbb{E}_{\mathcal{D}'}(Y) = 2k$ and

$\mathrm{Var}_{\mathcal{D}'}(Y) = 2k$, we use Chebyshev's inequality to claim that $\Pr(Y \le t) \le O(\frac{1}{k}) \le \frac{1}{2}$, and thus $\mathbb{E}_{\mathcal{D}'}(\delta^*(t)) \ge \frac{1}{2}$. Plugging these observations into Equation (B.1) yields that

$$\frac{\mathbb{E}_{\mathcal{D}}(\delta(t,1))}{\mathbb{E}_{\mathcal{D}}(\delta^*(t))} \ge \Omega(\log k) - \frac{\frac{1}{k}\cdot k}{\frac{1}{2}} = \Omega(\log k) - 2$$
$$= \Omega(\log k) = \Omega(\mu)$$

Applying Proposition B.1 to the distribution $\mathcal{D}$ yields a distribution $\hat{\mathcal{D}}$ with maximum distortion μ such that $\frac{\mathbb{E}_{\hat{\mathcal{D}}}(\mathrm{ALG})}{\mathbb{E}_{\hat{\mathcal{D}}}(\mathrm{OPT})} = \Omega(\mu)$, which completes the proof of the theorem.

C Poor Performance of Existing Algorithms

In this section, we show that some existing scheduling algorithms are not competitive in our setting.

Definition C.1. *When considering the running of an algorithm on some input, we denote the number of pending jobs in the algorithm at time t by $\delta(t)$. Similarly, we denote the number of pending jobs in the optimal solution at time t by $\delta^*(t)$. We also use the notation $\delta(t,x)$ to denote the number of pending jobs in the algorithm with remaining volume at least x.*

The following lemma is a restatement of the standard "bombardment" technique in flow-time scheduling.

Lemma C.1. *Consider a specific deterministic algorithm. If there exists an input for which $\delta(t,1) \ge c\cdot\delta^*(t)$ at some point in time t, then the algorithm is $\Omega(c)$-competitive.*

Proof. Suppose such an input I exists. Consider the modified input I' which behaves like I until time t, but from time t releases a job with processing time 1 every time unit for M time units.
The offline solution for this problem would behave like the optimal solution for I until time t, but would start working on the stream of jobs of processing time 1 from t onwards. At every time t' during the time interval $[t, t+M]$, the number of pending jobs in the offline solution is at most $\delta^*(t)+1$.
Meanwhile, the algorithm has no better option than working on the stream of jobs with processing time 1, which implies that $\delta(t') \ge \delta(t)+1$ for every $t' \in [t, t+M]$. Thus, as M tends to ∞, the ratio between the algorithm's cost and the offline solution's cost tends to $\frac{\delta(t)+1}{\delta^*(t)+1} \ge \frac{c}{2}$. This completes the proof.

C.1 Bad Case for SEPT The **SEPT** algorithm (shortest estimated processing time) would always choose to process a job from the minimum class of estimated processing time (preferring a partial job if possible).

Even without distortion, this algorithm has an unbounded competitive ratio. To see this, consider the following input for an arbitrarily large, even i:

1. For j from i down to $\frac{i}{2}$:
   (a) Release a job of processing time 2^j+1.
   (b) Wait 2^j time units.

Denote by t the time in which this input ends. At t, the algorithm **SEPT** would have $\frac{i}{2}+1$ pending jobs of remaining processing time 1, as it switches to each newly-released job upon its release. Thus, $\delta(t,1) \ge \frac{i}{2}$. However, the optimal solution could have $\delta^*(t)=1$ in the following way: follow **SEPT** until the job q of class $\frac{i}{2}$ is released, then use the remaining $2^{\frac{i}{2}}$ time units to finish all jobs except q (these jobs require only $\frac{i}{2}$ time to complete). Using Lemma C.1, and since i is arbitrarily large, the competitive ratio of **SEPT** is unbounded.

C.2 Bad Case for SR We now consider the special rule algorithm presented in [8], denoted by **SR**. In this algorithm, we again consider the classes of jobs. The algorithm always works on the lowest-class partial job q, until there exist two jobs such that one job is of class at most ℓ_q and the other is of class *strictly less* than ℓ_q. If such jobs exist, the algorithm chooses the one with minimal class and marks it as partial.

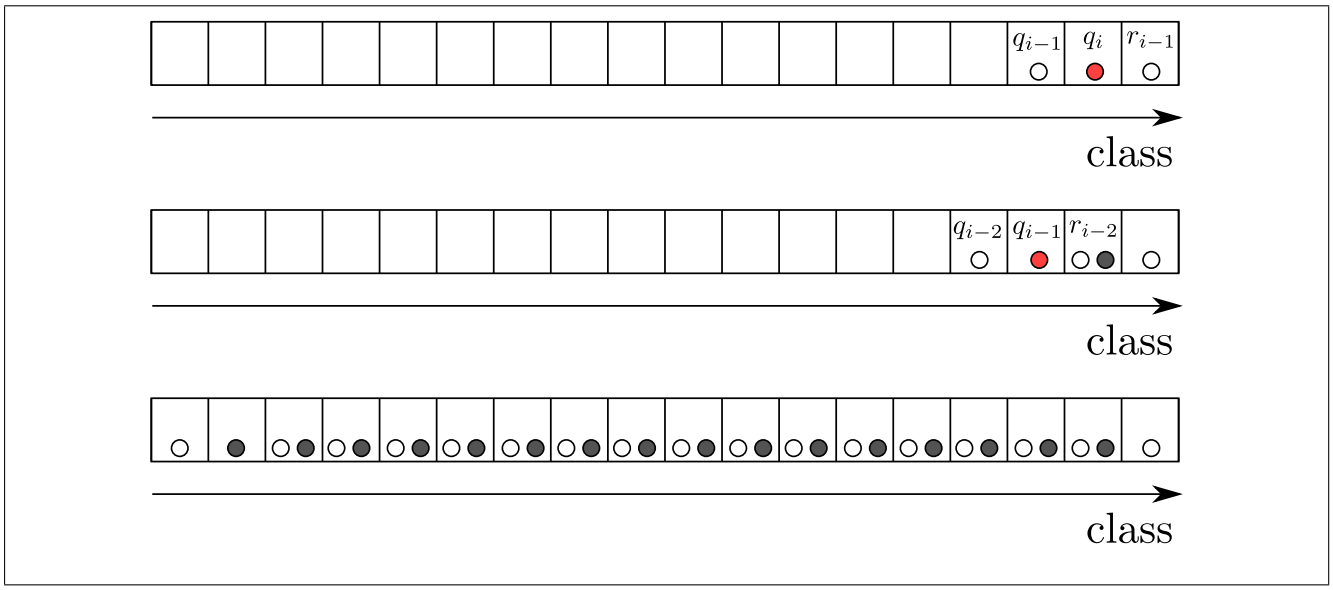


Figure 5: Constant Distortion in the **SR** Algorithm of [8]

We show that this algorithm has an unbounded competitive ratio even for a distortion parameter μ which is a moderate constant, specifically $\mu = 4$. In fact, we only require underestimations, and so we choose $\mu_2 = 1$ and $\mu_1 = 4$.

Let i be arbitrarily large. The input for which the algorithm fails is as follows:

1. Release a job q_i with estimated processing time 2^i and real processing time 2^{i+2} (class i, distortion of 4)
2. For j from $i-1$ down to 0:
   (a) Release a job q_j with estimated processing time 2^j and real processing time 2^{j+2} (class j, distortion of 4).
   (b) Release a job r_j with estimated processing time 2^{j+2} and real processing time 2^{j+2} (class $j+2$, no distortion).
   (c) Wait 2^{j+3} time units.

Figure 5 visualizes the bad input for **SR**. Initially, q_i arrives and becomes partial in the algorithm (a partial job appears as a red circle). Immediately afterwards, q_{i-1} and r_{i-1} are released, and are full in the algorithm. This state is shown in the first image of Figure 5.

Now, the input waits $8 \cdot 2^{i-1} = 2^{i+2}$ time units, during which the algorithm finishes q_i, and marks q_{i-1} as partial. Then, q_{i-2} and r_{i-2} are released. This state is shown in the second image of Figure 5 (the complete job q_i is shown as a gray circle).

As time progresses, the algorithm reaches the state in the third image of Figure 5, in which all jobs $\{r_j\}$ and the job q_0 are full and pending. Thus, $\delta(t,1) \geq i$.

Meanwhile, the optimal solution could, for every j, use the time spent by the algorithm on q_j to finish both q_{j-1} and r_{i-1}. Thus, the only living job in the optimal solution at t would be q_i, implying $\delta^*(t) = 1$. Using Lemma C.1 implies that the algorithm has a competitive ratio of $\Omega(i)$, and since i can be arbitrarily large, this competitive ratio is unbounded.

C.3 Bad Cases for the Robust Algorithm of [1] For every $\hat{\mu} > 1$, a $\hat{\mu}$-robust, $O(\hat{\mu}^2)$ competitive algorithm $\text{ALG}_{\hat{\mu}}$ was presented in [1]. We show two bad cases for the algorithm ALG_{μ}. First, we consider the case in which the distortion μ is much smaller than the distortion cap $\hat{\mu}$, and show that the algorithm is still $\Omega(\hat{\mu})$-competitive; In fact, we show this for the case that there is no distortion *at all*, i.e. $\mu = 1$. Second, we consider the case in

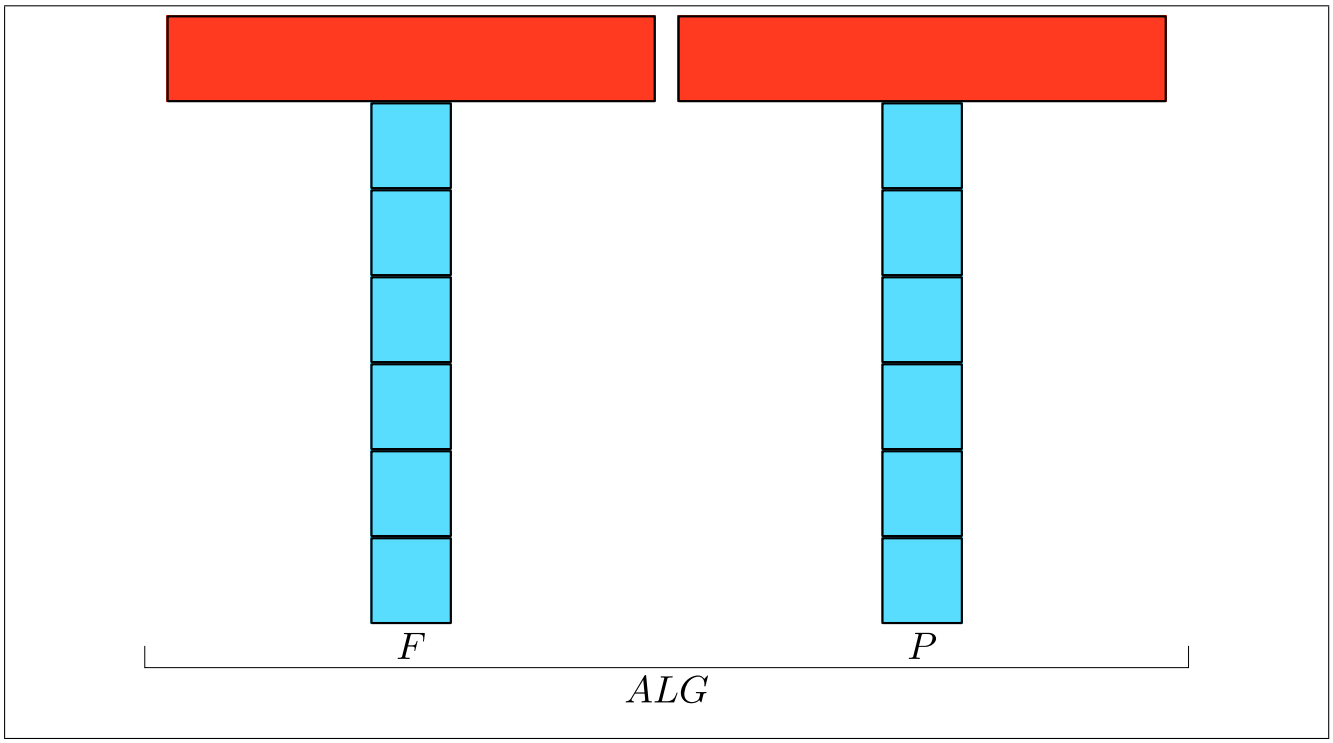


Figure 6: No Distortion in the Algorithm of [1]

which the distortion μ is slightly larger than $\hat{\mu}$ (specifically, $\mu = 4\hat{\mu}$), and show that the algorithm has unbounded competitiveness.

Bad case 1: $\mu = 1$ (no distortion). Assume $\hat{\mu}$ is an even integer for the sake of presentation. The input consists of releasing $\hat{\mu}$ jobs of size 1 and 2 jobs of size $\frac{\hat{\mu}}{2}$ at time 0, then waiting for $\hat{\mu}$ time units.

The state of the algorithm immediately after the release of the jobs is visualized in Figure 6. This visualization follows the description in [1] (i.e. each job is a rectangle whose width is the job's remaining volume). During the waiting time of $\hat{\mu}$ time units, the algorithm would work and finish the two jobs of volume $\frac{\hat{\mu}}{2}$, while the optimal solution could finish all jobs of volume 1. Thus, at $t = \hat{\mu}$ it holds that $\frac{\delta(t,1)}{\delta^*(t)} = \Omega(\hat{\mu})$, and thus Lemma C.1 implies that the algorithm is $\Omega(\hat{\mu})$-competitive even when there is no distortion.

Bad case 2: $\mu = 4\hat{\mu}$. Assume $\hat{\mu} = 2^m$ for some integer m. We construct a somewhat similar adversary to that previously described for **SR** with distortion $4\hat{\mu} = 2^{m+2}$ and show that the algorithm has unbounded competitive ratio on this input

We choose an arbitrarily large integer i. The adversary performs the following actions:

1. Release a job d of arbitrary volume.
2. Release a job q_i such that $\tilde{p}_{q_i} = 2^i$, $p_{q_i} = 2^{i+m+2}$.
3. For j from $i-1$ down to 1:
   (a) Release a job r_j such that $\tilde{p}_{r_j} = p_{r_j} = 2^{j+m}$ (no distortion).
   (b) Release a job q_j such that $\tilde{p}_{q_j} = 2^j$, $p_{q_j} = 2^{j+m+2}$ (distortion $4\hat{\mu}$)
   (c) Wait $3 \cdot 2^{j+m}$ time units.

Denote the time in which the adversary ends by t. We consider a visual representation of the algorithm's operation given in Figure 7 (which again follows the description in [1]). Initially, d is released and immediately moved by

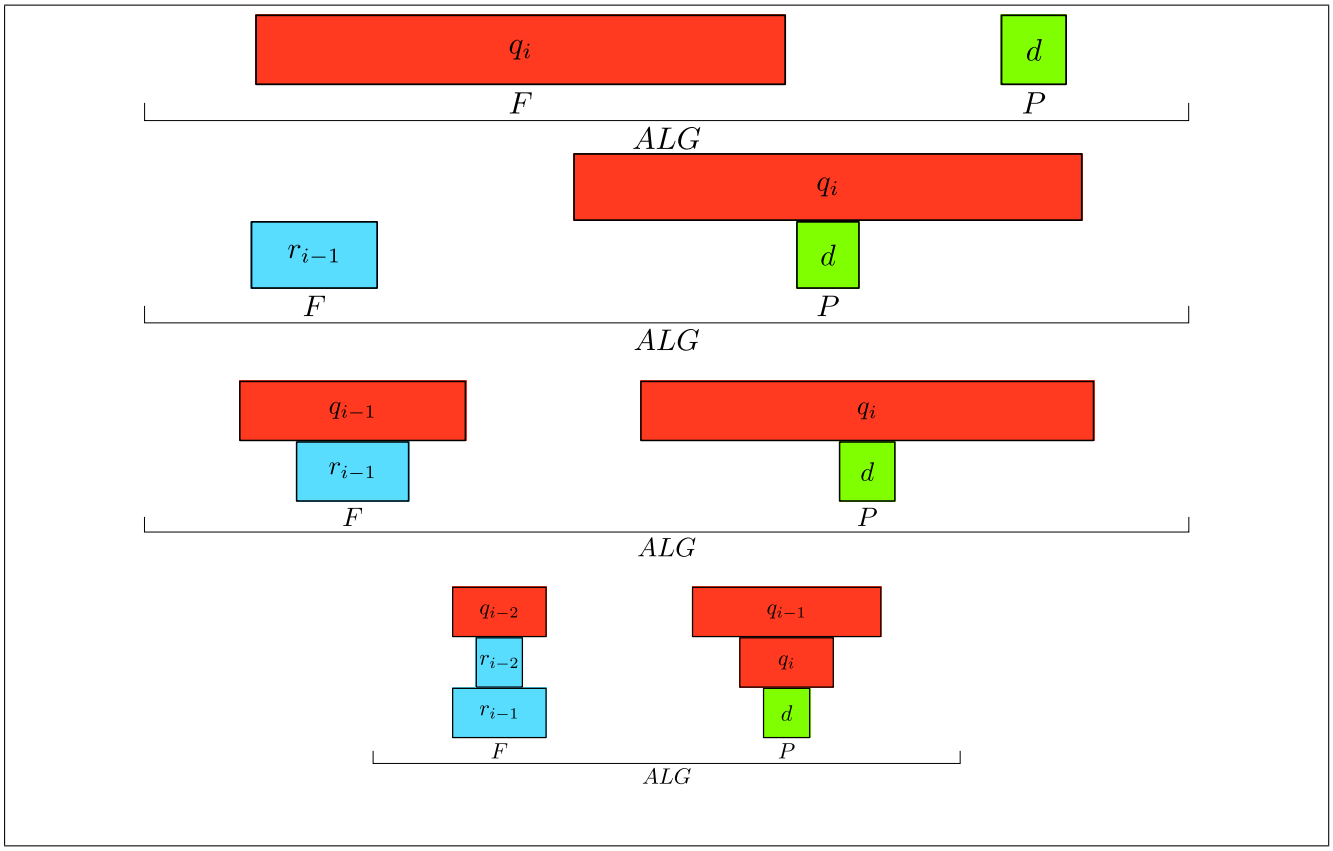


Figure 7: Excessive Distortion in the Algorithm of [1]

the algorithm to the partial bin P. q_i is released immediately afterwards, and is put in the full bin F. The first image of Figure 7 shows the state at this point.

Now, r_{i-1} is released into F, and swaps with q_i (as its estimate is larger by a factor of $\hat{\mu}$). The job q_i immediately moves to P. The current state is shown in the second image of Figure 7. Now, the job q_{i-1} is released to the top of F, as shown in the third image of Figure 7.

Now, the adversary waits $3 \cdot 2^{i-1+m}$ time, during which the algorithm works on q_i (and doesn't complete it). Meanwhile, the optimal solution would finish both q_{i-1} and r_{i-1}. Afterwards, the adversary releases r_{i-2} (which swaps with q_{i-1}, causing it to move to P) and then q_{i-2}. This state is shown in the fourth image of Figure 7. Now, the adversary waits for $3 \cdot 2^{i-2+m}$ time, during which the algorithm works on q_{i-1} and doesn't finish, and the adversary finishes q_{i-2} and r_{i-2}.

As this process continues, the algorithm will have all $2i$ jobs alive at t (with at least one using of volume remaining) and thus $\delta(t, 1) \geq 2i$. Meanwhile, the optimal solution would only have two jobs pending at t (d and q_i), and thus $\delta^*(t) \leq 2$. Lemma C.1 implies that the algorithm is $\Omega(i)$-competitive. Since i is arbitrarily large, this implies that the algorithm $\mathrm{ALG}_{\hat{\mu}}$ has unbounded competitive ratio on inputs with distortion $4\hat{\mu}$.