Distortion-Oblivious Algorithms for Scheduling on Multiple Machines

Yossi Azar
Tel Aviv University, Israel

Eldad Peretz
Tel Aviv University, Israel

Noam Touitou
Tel Aviv University, Israel
Amazon, Tel Aviv, Israel[1]

Abstract

We consider the classic online problem of scheduling on multiple machines to minimize total flow time and total stretch where the input consists of estimates on the processing time provided for each job once released. The performance of such algorithms should depend on μ, the error in the estimates of the processing time for that instance (such an algorithm is called a distortion oblivious algorithm). Previously, a distortion oblivious algorithm to minimize flow time was provided only for a single machine. In this paper we extend the work to multiple machines and also consider the total stretch objective. In particular, we design a non-migrative distortion oblivious algorithm to minimize total flow time with a competitive ratio of $O(\mu \log P)$, where P is the ratio between the maximum to minimum processing time. We show that with immediate-dispatching one cannot achieve a competitive ratio which is a function of μ and P; moreover, a competitive ratio which is sub-polynomial in the number of jobs is also impossible. We also present the first distortion-oblivious algorithm for minimizing the stretch time, both on a single and on multiple machines. The competitive ratio of these algorithms are $O(\mu^2)$ which is optimal as we also prove a matching $\Omega(\mu^2)$ lower bound.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online, Scheduling, Predictions, Stretch, Flow Time

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.16

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

1 Introduction

We consider the online scheduling problem on multiple parallel machines, where n jobs arrive over time and the completion of a job requires processing time on the $m \geq 1$ machines. The machines are parallel, so at at any given time a job can be executed on a single machine only. The goal of a scheduling algorithm is to minimize a certain objective function. In this paper, we consider both the *total flow time* objective, which is the sum of the jobs flow-time (time from release to completion), and the *total stretch* objective, which normalizes the flow time of each job by the required processing time for that job. We consider the variant of the problem in which preemption is allowed (i.e., the algorithm is allowed to halt and resume the processing of jobs as desired).

In **classic scheduling**, the processing times of the jobs are exactly known at the job release time. In a single machine setting, it is well known that the algorithm *SRPT* (shortest remaining processing time) [23] is 1-competitive. It has been shown that on multiple machines

[1] This work was done prior to joining Amazon.

the problem is more difficult and depends on the ratio between the largest processing time of a job to the minimum one (denoted by P). Specifically, every online algorithm is $\Omega(\log P)$ competitive. In addition, $SRPT$ algorithm is an optimal online scheduling algorithm which achieves a tight competitive ratio of $O(\log P)$ [15].

However, in practice usually the assumption that the job processing time is known at the job release time does not hold. For example, in computer programs scheduling it is very unlikely to know the job's exact processing time in advance.

Scheduling with estimates, introduced in [4], addresses this lack of processing-time knowledge. In this setting, upon job release the algorithm is provided with an *estimate* of the job's processing time, which might be inaccurate up to a multiplicative error of μ; this parameter μ is called the *distortion* of the input. Naturally, as μ increases, the best achievable competitive ratio becomes worse. A possible result in this model is a *robust family of algorithms* $\{ALG_\mu\}$, such that ALG_μ is tailored to appropriately handle inputs of distortion at most μ. To use such a family, one needs to know the distortion of the input μ in advance to choose the correct algorithm. However, in practical settings, knowing the distortion in advance is infeasible; moreover, guessing the distortion could yield unbounded competitiveness (see [5]). Thus, a preferable result is a single algorithm (rather than a family) that handles any input, while achieving a competitive ratio as a function of the distortion of that input. Such an algorithm is called *distortion-oblivious.* In our paper, we define the parameter P to be the maximum ratio between *estimated* job sizes. We focus on presenting distortion-oblivious algorithms for scheduling with estimates.

In this paper, we consider preemptive **scheduling on multiple machines**. In such scheduling problems, certain properties are desired in a good algorithm. For example, such a property is being *non-migrative*, i.e., once the algorithm decides to process a job on a machine, the job can never be processed on a different machine. Another, stronger property is *immediate dispatching*, in which a job must also be assigned to a machine immediately upon its release.

1.1 Our Results

In this paper, we present the following distortion-oblivious algorithms and lower bounds for scheduling with estimates.

1. We show a non-migrative algorithm with a competitive ratio of $O(\mu \log P)$ with respect to the total flow time objective compared to optimal migrative algorithm (Theorem 2). If migration is allowed we show using similar analysis that lowest class first also achieves a competitive ratio of $O(\min(\mu \log P, \mu \log \mu + \mu \log \frac{n}{m}))$ (Corollary 16).
2. We present a tight non-migrative algorithm for minimizing total stretch on multiple machines with a competitive ratio of $O(\mu^2)$. (Theorem 6)
3. We present a matching lower bound for minimizing total stretch on multiple machines (that even allows migration) of $\Omega(\mu^2)$. (Theorem 11)

In particular, Results 2, 3 also apply for a single machine, and are the first known results in this setting. Therefore, in this paper we completely solve the problem of distortion-oblivious algorithms to minimize stretch time.

We also consider the immediate dispatching property for scheduling with estimates. Here, we show that there exists no algorithm of competitiveness sub-polynomial in the number of jobs, even for arbitrarily-small μ and P; in particular, there is no algorithm with competitive ratio as a function of only μ and P. This lower bound applies to both total flow time and total stretch, and holds even when randomization is allowed. Details appear in Section 5.

1.2 Related Work

A similar setting to the robust setting is nonclairvoyant scheduling, studied in [21, 8, 7, 11, 10]. In this setting, the size of each job is completely unknown at release time, and is only revealed to the algorithm upon completion of the job. In fact, this non-clairvoyant setting is a special case of our model in which the distortion μ is equal to P (i.e., the predicted processing time is meaningless). In this nonclairvoyant model, for multiple machines, the best known result for minimizing flow time is $O(\log n \log \frac{n}{m})$ [8].

In the classic scheduling setting, the exact processing times of jobs are known at release time; this classic model is a special case of the robust setting in which $\mu = 1$. In this setting, [15] presented an $O(\log P)$ algorithm and a matching lower bound of $\Omega(\log P)$. It is also known that the same algorithm achieves an upper bound of $O(\log(n/m))$; without migration, [3] presented an $O(\log n)$-competitive algorithm. A result of similar competitiveness was introduced by [2] for the immediate dispatching variant of this problem. For minimizing stretch, an $O(1)$ non-migrative competitive algorithm is known [9].

Scheduling with distortion falls within the field of algorithms with predictions. In this model, an online algorithm is given somewhat-accurate predictions of the incoming online input. A good algorithm should be able to benefit from these predictions to the degree to which they are precise; its competitive ratio would thus be a function of the predictions' accuracy. Many problems related to scheduling have been addressed using algorithms with predictions; for example, see [22, 13, 19, 12, 6, 16]. Algorithms with predictions have also been used for many other online problems [17, 18, 20]. Additional papers on algorithms with predictions can be found in [1].

1.3 Paper Organization

The non-migrative algorithm for minimizing total flow time is presented and analyzed in Section 3. The non-migrative algorithm for minimizing total stretch is presented and analyzed in Section 4. In Section 5 we show lower bound for immediate dispatching distortion oblivious algorithms, for both the flow time and total stretch objectives. In Appendix A, we show additional results for the total flow time objective; specifically, we analyze the LCF algorithm and give tight examples for the shown algorithms.

2 Preliminaries

In our scheduling problem, jobs are released over time. There are m parallel machines; that means at any given time a job can be processed only on one of the m machines. Each job q must be processed for exactly p_q time (p_q is called the *processing time* of q). For a job q, we denote by r_q its release time and upon the job release time the algorithm gets the job estimated processing time $\tilde{p_q}$; the actual processing time of the job is unknown and revealed only at the job completion time c_q.

We define $\mu_1 = \max_j \frac{p_j}{\tilde{p_j}}$ as the maximum underestimation factor of a job in the input. Similarly, also define $\mu_2 = \max_j \frac{\tilde{p_j}}{p_j}$ as the maximum overestimation factor of a job in the input. Hence, for any job q:

$$\frac{\tilde{p_q}}{\mu_2} \leq p_q \leq \tilde{p_q} \cdot \mu_1$$

Finally, we define the distortion parameter $\mu := \mu_1 \cdot \mu_2$. It is natural to assume that $\mu_1, \mu_2 \geq 1$, although it is not required for our proofs.

Let P be the ratio between the maximal job estimated size and the minimal job estimated size in the input, $P = \frac{\max_q \tilde{p_q}}{\min_q \tilde{p_q}}$.

Note that since P is a function of the input the algorithm has no prior knowledge on P. For the same reason, unless the algorithm is not distortion oblivious it also has no knowledge on the parameters μ_1, μ_2 and μ.

Objectives. In our paper we discuss 2 objectives: total flow time and total stretch. The total flow time objective is one of the most basic performance measures in multiprocessor scheduling problems, and is defined as the overall time the jobs are spending in the system. This includes the delay of waiting for processing as well as the actual processing time. Formally, using the defined notations the total flow time objective is:

$$F = \sum_q c_q - r_q$$

Another natural objective is the total stretch objective. The total stretch objective is defined as follows:

$$W = \sum_q \frac{c_q - r_q}{p_q}.$$

To refer to the objective of an algorithm A, we use the superscript A, e.g., F^A or W^A. Unlike the classic setting, in scheduling with estimates the total stretch objective is *not* a special case of the total weighted time objective since the algorithm does not know the actual processing times of the alive jobs and therefore does not know their weights.

Throughout the paper, we often use the following definition of a job class:

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

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 and the definition does not depend on the algorithm (either it is an algorithm or the optimal solution the job classes are defined the same).

Notations. In the paper we use notations that have been defined by previous work:

- $V^A(t)$ is the remaining time of alive jobs for algorithm A at time t
- $\delta^A(t)$ is the number of alive jobs for algorithm A at time t.
- $\gamma^A(t)$ is the number of non-idle machines for algorithm A at time t.

Where the algorithm is clear from context, we sometimes omit the superscript for these definitions. In addition, for a function f (e.g., V or δ) we define the following notations:

- $\Delta f(t) = f^A(t) - f^{OPT}(t)$, the difference at f value between the algorithm A and the optimal solution at time t.
- $f_{=k}(t)$ is the value of f at time t for class k.
- $f_{\le k}(t)$ is the value of f at time t over all jobs of class at most k. Similarly, $f_{\ge k}(t)$ is the value of f at time t over all jobs of class at least k

We also use * to denote the optimal solution. For example, $V^*(t)$ is the remaining time of alive jobs for the optimal solution at time t.

3 Minimizing Flow Time

In this section, we describe and analyze a distortion-oblivious algorithm for minimizing total flow time for the non-migrative setting with competitive ratio of $O(\mu \log P)$ against even an migrative optimal algorithm. Note that when μ is a constant this is the best known upper bound.

3.1 Algorithm for the Non-Migrative Setting

The algorithm maintains a pool of jobs that are released and have not been processed at all. In addition, it maintains at every machine a stack of alive jobs that have already been processed by that machine. At any time, each machine processes the job at the top of its stack. The algorithm handles two events: job release and job completion.

At job release, the algorithm checks whether there exists a machine that is empty or currently processing a job of higher class. If such a machine exists, the algorithm places the job at the top of that machine's stack. Otherwise, the job is added to the pool.

Upon job completion, the algorithm pops the completed job from its associated machine stack. Then, it observes the job q' currently at the top of that stack. If the lowest-class job q'' in the pool has a lower class than q' (or if q' does not exist) then q'' is moved to the top of the stack of that machine.

The online algorithm is given as Algorithm 1.

Algorithm 1 Distortion Oblivious Non Migrative Scheduling Algorithm.

```
1  Event Function UponJobRelease(q)
2      if Exists an idle machine or a machine that currently processes a job of class
        higher than $\ell_q$ then
3          Insert $q$ to the top of that machine stack.
4      else
5          Insert $q$ to the pool.
6  Event Function UponJobCompletion(q)
7      Denote by $m_j$ the machine $q$ was processed on.
8      Pop $q$ from the stack of $m_j$, and let $q'$ be the next job in that stack.
9      if the job with lowest class in the pool $q''$ has class strictly less than that of $q'$ or
        $q'$ does not exist then
10         Remove $q''$ from the pool and insert it to the top of the stack of $m_j$.
```

The following theorem (Theorem 2) shows that Algorithm 1 has a bounded competitive ratio against an optimal offline solution for the input; in Appendix A.2, we show that Theorem 2 is in fact tight for $m \geq 2$.

▶ **Theorem 2.** *Algorithm 1 is $O(\mu \log P)$-competitive for inputs with distortion μ.*

3.2 Proof of Theorem 2

A useful and classic concept that we use in the proof, is called *local competitiveness*. An algorithm is locally competitive if at every point in time t, the number of living jobs in the algorithm at t is at most some factor from that of the optimal solution at t. A classic observation is that a locally c-competitive algorithm is in particular (globally) c-competitive; this observation results simply from integrating over time. Observation 3 states this formally.

▶ **Observation 3.** *If for an input I it holds that at every time t we have $\delta^A(t) \leq c(I) \cdot \delta^*(t)$ then the algorithm A is $c(I)$-competitive with respect to total flow time on input I.*

Note that at any time every machine stack has at most one job of each class. There are $\lceil \log P \rceil$ classes, therefore at any time the number of partial jobs is at most $m \log P$. Also note that if at time t there is an idle machine (i.e., $\gamma(t) < m$) then the pool is empty and the the total number of alive jobs of the algorithm at time t is $\delta^{ALG}(t) \leq m \log P$.

We define T to be the set of times t that $\gamma^{ALG}(t) = m$; that is, the set of times in which all machines are busy.

We start by proving the following lemma, that for every k bounds the total remaining processing time difference on jobs of class at most k between the algorithm and the optimal solution.

▶ **Lemma 4.** *If $t \in T$ then $\Delta_{\leq k} V(t) \leq m\mu_1 \cdot 2^{k+2}$.*

Proof. Let t_0 be the earliest time such that $[t_0, t) \subset T$. We define $t_k \in [t_0, t)$ as the latest time in $[t_0, t)$ that the algorithm has processed a job of class greater than k (if no such job was processed during this interval then $t_k = t_0$). At time t_k we know that there are no pending jobs of class at most k in the pool. Therefore, all jobs of class at most k are already assigned to a machine. Since every machine process at most one job of any class, there are at most m jobs of any class at most k in the system. The worst case actual processing time of a job of class i is $m\mu_1 2^{i+1}$, thus

$$\Delta_{\leq k} V(t_k) \leq \sum_{i=1}^{k} m\mu_1 2^{i+1} \leq m\mu_1 2^{k+2}.$$

In $[t_k, t) \subset T$ we processed only jobs of class at most k, implies that

$$\Delta_{\leq k} V(t) \leq \Delta_{\leq k} V(t_k) \leq m\mu_1 2^{k+2}.$$

Note that arrival of new jobs adds to both V^* and V^{ALG} the same amount and therefore does not affect the proof. ◀

▶ **Lemma 5.** *If $t \in T$ then $\delta^{ALG}(t) \leq (\mu + 3)\gamma^{ALG}(t) \log P + 2\mu\delta^*(t)$*

Proof.

$$\begin{aligned}
\delta^{ALG}(t) &= \sum_{i=k_{min}}^{k_{max}} \delta^{ALG}_{=i}(t) \\
&\leq \sum_{i=k_{min}}^{k_{max}} \left(\frac{V^{ALG}_{=i}(t)}{2^i/\mu_2} + m \right) \\
&\leq \sum_{i=k_{min}}^{k_{max}} \frac{V^*_{=i}(t) + \Delta V_{=i}(t)}{2^i/\mu_2} + m \log P \\
&\leq \sum_{i=k_{min}}^{k_{max}} \frac{\delta^*_{=i}(t)\mu_1 2^{i+1}}{2^i/\mu_2} + \sum_{i=k_{min}}^{k_{max}} \frac{\Delta V_{\leq i}(t) - \Delta V_{\leq i-1}(t)}{2^i/\mu_2} + \gamma^{ALG}(t) \log P \\
&\leq 2\mu\delta^*(t) + \sum_{i=k_{min}}^{k_{max}} \frac{\Delta V_{\leq i}(t)}{2^{i+1}/\mu_2} + \frac{\Delta V_{\leq k_{max}}(t)}{2^{k_{max}+1}/\mu_2} + \gamma^{ALG}(t) \log P \\
&\leq 2\mu\delta^*(t) + \gamma^{ALG}(t)\mu \log P + 2\mu\gamma^{ALG}(t) + \gamma^{ALG}(t) \log P \\
&\leq (\mu + 3)\gamma^{ALG}(t) \log P + 2\mu\delta^*(t)
\end{aligned}$$

where the first inequality is since there are at most m partial jobs at each class. The third inequality is since $t \in T$ thus $\gamma^{ALG}(t)$ equals m. The fourth inequality is since $\delta^*(t) = \sum_{i=k_{min}}^{k_{max}} \delta^*(t)$. The fifth inequality is applying Lemma 4. ◀

Now we turn to prove Theorem 2:

Proof of Theorem 2. Using the analysis above, we can now bound the total flow time of the algorithm.

$$\begin{aligned}
F^{ALG} &= \int_t \delta^{ALG}(t)dt \\
&= \int_{t\in T} \delta^{ALG}(t)dt + \int_{t\notin T} \delta^{ALG}(t)dt \\
&\le \int_{t\in T} (\mu+3)\gamma^{ALG}(t)\log(P) + 2\mu\delta^*(t)dt + \int_{t\notin T} \gamma^{ALG}(t)\log(P)dt \\
&\le (\mu+3)\log P \int_t \gamma^{ALG}(t)dt + 2\mu \int_t \delta^*(t)dt \\
&\le (\mu+3)\log P \cdot F^* + 2\mu \cdot F^* \\
&= O(\mu \log P)F^*
\end{aligned}$$

where the first inequality follows by applying Lemma 5. The last inequality follows since $F^* = \int_t \delta^*(t)dt$ and $\int_t \gamma^{ALG}(t)dt \le F^*$. ◀

4 Minimizing Total Stretch

In this section, we study the distortion-oblivious, non-migrative Algorithm 1 for minimizing total stretch. We show that Algorithm 1 achieves the best possible competitive ratio for non-migrative algorithms. We show that Algorithm 1 competitive ratio is $O(\mu^2)$ against even an migrative optimal algorithm (Theorem 6). In addition, we also prove a matching lower bound of $\Omega(\mu^2)$ even for the migrative case. Those results concludes that Algorithm 1 is optimal (Theorem 11) for minimizing the total stretch objective in the online setting.

The weight of a job q is inversely proportional to q's (actual) processing time; we denote this weight by $w_q := \frac{1}{p_q}$. Similarly, for a set of jobs Q we define w_Q as the sum of the weights $w_Q := \sum_{q\in Q} w_q$.

In addition, we define $W^A(t)$ to be the total weight of the currently-alive jobs of an algorithm A at time t (and omit A whenever the algorithm is clear from context). We also use a subscript predicate to restrict this notation to specific classes; for example, $W_{=i}(t)$ is the total weight of living jobs of class exactly i in the algorithm at time t.

4.1 Non-Migrative $O(\mu^2)$ Scheduling on Parallel Machines

In this subsection, we prove the following theorem.

▶ **Theorem 6.** *Algorithm 1 is $O(\mu^2)$-competitive for inputs with distortion μ to minimize total stretch.*

Since stretch scheduling involves weights, here local competitiveness means that at every time t, the total living weight in the algorithm is bounded by the total living weight in the optimal solution. Through integration over time, it is easy to see that local-competitiveness implies competitiveness, as stated in Observation 7.

▶ **Observation 7.** *If for any input I, at any time t, $W^A(t) \leq c(I) \cdot W^*(t)$ then the algorithm A is $c(I)$-competitive with respect to total stretch.*

We start with the following notation:

- Let $c_j(t)$ be the currently processed job of machine j at time t (i.e., the job at the top of machine j stack).
- Let $R(t) = \{c_j(t) | j \in [m]\}$ is the set of jobs being processed at time t.
- Let $S_j(t)$ denote the the set of alive jobs that are on machine j stack, excluding the currently processed job $c_j(t)$. Let $S(t) = \bigcup_j S_j(t)$ the set of alive jobs in one of the machines stack and not currently processed.
- Let $P(t)$ denote the set of alive jobs in the pool at time t.

▶ **Observation 8.** *For any scheduling algorithm A, $\int_t \sum_{x \in R^A(t)} w_x = n$ from the definition of the stretch objective, since every job q is executed exactly p_q time in the algorithm. Therefore, both OPT and Algorithm 1 has total stretch of at least n ($W^* \geq n$).*

We will split the proof into two lemmas. The first lemma (Lemma 9) will show that the total stretch added from jobs being in the machines stacks is bounded by $2n\mu$ and therefore from Observation 8 we get that this part is $O(\mu)$ competitive with OPT. The second lemma (Lemma 10) shows that the rest of the jobs time in the system add at most $4\mu^2 W^* + 12\mu^2 n$ stretch, which is $O(\mu^2)$ competitive with OPT. From this two lemmas, Theorem 6 follows.

▶ **Lemma 9.** $\int_t \sum_{x \in S(t)} w_x \leq 2n\mu$.

Proof. At every machine stack, there is at most one job of each class. Every job $x \in S_j(t)$ is of class strictly greater than of $c_j(t)$.

$$w_{S_j(t)} = \sum_{x \in S_j(t)} w_x \leq \sum_{i=\ell_{c_j(t)}}^{k_{max}} \frac{\mu_2}{2^i} \leq \frac{\mu_2}{2^{\ell_{c_j(t)}}}.$$

The weight of the currently running job of machine j is $w_{c_j(t)} \geq \frac{1}{2^{\ell_{c_j(t)}+1}\mu_1}$. Hence,

$$w_{S_j(t)} \leq 2\mu w_{c_j(t)}.$$

Therefore, $w_{S(t)} \leq 2\mu \cdot w_{R(t)}$ and since $\int_t w_{R(t)} = n$ we get $w_{S(t)} \leq 2\mu n$ that concludes the proof. ◀

Let $V^P_{=i}(t)$ denote the volume of jobs in the pool of class i, $V^P_{\leq i}(t) = \sum_{z \leq i} V^P_{=z}(t)$ and $\Delta V^P_{\leq i}(t) = V^P_{\leq i}(t) - V^*_{\leq i}(t)$. Let $\rho(t)$ denote $\max_{x \in R(t)} \ell_x$, the maximal class of a running job at time t. From algorithm definition, all jobs in the pool are of class at least $\rho(t)$, therefore $V^P_{<\rho(t)}(t) = 0$. We also define $\alpha(t)$ to be the largest class of an alive job. We now turn to prove the lemma that bounds the weight of jobs in the pool.

▶ **Lemma 10.** $\sum_{x \in P(t)} w_x \leq 6\mu^2 W^*(t) + 12\mu^2 \sum_{x \in R(t)} w_x$.

Proof. Note that in case $\gamma(t) < m$, then the pool is empty and the lemma is trivially true; since the left hand side is 0. Denote by C_i the set of jobs of class i. The number of jobs in $C_i \cap P(t)$ is upper bounded by $\frac{\mu_2 V^P_{=i}(t)}{2^i}$ since the jobs in the pool aren't processed yet. Each job has a weight at most $\frac{\mu_2}{2^i}$. Hence we get that

$$\begin{aligned}\sum_{i=\rho(t)}^{\alpha(t)} \sum_{x\in C_i\cap P(t)} w_x \leq& \sum_{i=\rho(t)}^{\alpha(t)} \mu_2 \frac{V^P_{=i}(t)}{2^i}\cdot\frac{\mu_2}{2^i}\\ =&\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{V^P_{=i}(t)}{2^i\cdot 2^i}\\ \leq&\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{V^*_{=i}(t)+\Delta V^P_{=i}(t)}{2^i\cdot 2^i}\\ \leq&\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{V^*_{=i}(t)}{2^i\cdot 2^i} + \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{=i}(t)}{2^i\cdot 2^i}.\end{aligned}$$

First, we bound $\mu_2^2\sum_{i=\rho(t)}^{\alpha(t)} \frac{V^*_{=i}(t)}{2^i\cdot 2^i}$. At time t, OPT has at least $\frac{V^*_{=i}(t)}{2^{i+1}\mu_1}$ jobs of class i with weight at least $\frac{1}{2^{i+1}\mu_1}$. Therefore:

$$W^*_{=i}(t) \geq \frac{V^*_{=i}(t)}{2^{i+1}\mu_1}\cdot\frac{1}{2^{i+1}\mu_1}$$

which is equivalent to

$$\mu_2^2\cdot\frac{V^*_{=i}(t)}{2^i\cdot 2^i} \leq 4\mu^2 W^*_{=i}(t)$$

and concludes that

$$\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{V^*_{=i}(t)}{2^i\cdot 2^i} \leq 4\mu^2 W^*(t).$$

It remains to show that $\mu_2^2\sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{=i}(t)}{2^i\cdot 2^i} \leq 6\mu^2\sum_{x\in R(t)} w_x + 2\mu^2 W^*(t)$:

$$\begin{aligned}\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{=i}(t)}{2^i\cdot 2^i} \leq& \mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{\leq i}(t)-\Delta V^P_{\leq i-1}(t)}{2^i\cdot 2^i}\\ =&\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{\leq i}(t)}{2^{i+1}\cdot 2^i} + \mu_2^2\frac{\Delta V_{\leq\alpha(t)}(t)}{2^{\alpha(t)+1}\cdot 2^{\alpha(t)}} - \mu_2^2\frac{\Delta V^P_{\leq\rho(t)-1}(t)}{2^{\rho(t)+1}\cdot 2^{\rho(t)}}\\ =&\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{\leq i}(t)}{2^{i+1}\cdot 2^i} + \mu_2^2\frac{\Delta V_{\leq\alpha(t)}(t)}{2^{\alpha(t)+1}\cdot 2^{\alpha(t)}} + \mu_2^2\frac{V^*_{\leq\rho(t)-1}(t)}{2^{\rho(t)+1}\cdot 2^{\rho(t)}}\\ \leq&\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{\leq i}(t)}{2^{i+1}\cdot 2^i} + \mu_2^2\frac{\Delta V_{\leq\alpha(t)}(t)}{2^{\alpha(t)+1}\cdot 2^{\alpha(t)}} + \mu_2^2\frac{W^*_{\leq\rho(t)-1}(t)\cdot\mu_1^2\cdot 2^{2\rho(t)+2}}{2^{\rho(t)+1}\cdot 2^{\rho(t)}}\\ \leq&\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{(m-1)\mu_1}{2^i} + \mu_2^2\frac{4(m-1)\mu_1}{2^{\alpha(t)}} + 2\mu^2 W^*(t)\\ \leq&6\mu_2^2\mu_1(m-1)\cdot\frac{1}{2^{\rho(t)}} + 2\mu^2 W^*(t)\\ \leq&6m\mu^2\sum_{x\in R(t)} w_x + 2\mu^2 W^*(t)\end{aligned}$$

where the second inequality is since $\delta^*_{\leq\rho(t)}(t) \geq \frac{V^*_{\leq\rho(t)}}{\mu_1 \cdot 2^{\rho(t)+1}}$ and the minimal weight of a job of class at most $\rho(t)$ is $\frac{1}{\mu_1 \cdot 2^{\rho(t)+1}}$. The first two terms of the third inequality are according to Lemma 4 of previous section (and the fact that $\Delta V^P_{\leq i}(t) \leq \Delta V_{\leq i}(t)$). The right most term of the third inequality is since $W^*_{\leq\rho(t)}(t) \leq W^*(t)$. The last inequality is since for every $x \in R(t)$, $w_x \geq \frac{1}{2^{\rho}(t)\mu_1}$ we get that $6\mu_2^2\mu_1(m-1)\frac{1}{2^{\rho(t)}} \leq 6m\mu^2 \sum_{x\in R(t)} w_x$. We showed that $\mu_2^2 \sum_{i=\rho(t)}^{\alpha(t)} \frac{\Delta V^P_{=i}(t)}{2^i \cdot 2^i} \leq 6\mu^2 \sum_{x\in R(t)} w_x + 2\mu^2 W^*(t)$ which concludes the proof. ◀

Now we can prove the algorithm is $O(\mu^2)$ competitive:

Proof of Theorem 6.

$$\begin{aligned}\int_t \sum_{x\in Jobs(t)} w_x &= \int_t \sum_{x\in S(t)} w_x + \int_t \sum_{x\in P(t)} w_x \\ &\leq 3n\mu + \int_t 6\mu^2 W^*(t) + 6\mu^2 \sum_{x\in R(t)} w_x \\ &\leq 3n\mu + 6\mu^2 W^* + 6\mu^2 n \\ &\leq 15\mu^2 W^*\end{aligned}$$

where the second inequality is since $\int_t \sum_{x\in R(t)} w_x = n$ (Observation 8) and the last inequality is since $n \leq W^*$ (Observation 8). ◀

4.2 Total Stretch Lower Bound

We show $\Omega(\mu^2)$ lower bound for online distortion oblivious scheduling on multiple machines even if job migrations is allowed. This lower bound proves that the online algorithm presented in this section is optimal for total stretch minimization.

▶ **Theorem 11.** *For every number of parallel machines m and for every constant distortion μ, every deterministic online algorithm ALG to minimize total stretch time has a competitive ratio of $\Omega(\mu^2)$.*

Proof. Without loss of generality the algorithm is non-idle; since every algorithm with idle-time can be transformed to a non-idle algorithm by simply processing a arbitrary job during the idle time. Also assume that $\mu > 2$. Let $t = \frac{\mu^5}{2}$. We analyze the local competitive ratio of the algorithm at time t and then extend the result to (global) competitive ratio.

At time 0 release $m \cdot \mu^4$ jobs of estimated size 1 (their actual size is in $[1, \mu]$). For every job q denote by x_q the amount of time ALG processes job q until time t. For every job q, define the actual processing time of q to be:

$$p_j = \min(\max(1, x_j + \epsilon), \mu)$$

where $\epsilon = \frac{1}{\mu}$. Note that the sum of the jobs processing time is at least $m \cdot t$, since we assume the algorithm is non-idle and there is enough volume to be processed. Denote:

1. D is the set of jobs that $p_j = \mu$.
2. F is the set of jobs that $x_j \leq 1 - \epsilon$. In particular for all $j \in F$, $p_j = 1$.
3. P is the set of all the other jobs.

The above implies that:

$$|D| + |F| + |P| = m \cdot \mu^4.$$

OPT processes the jobs with *Shortest Time First*. Therefore, OPT finishes all the jobs of size at most $\frac{\mu}{2}$ (in particular, all the jobs in F). Note that the total remaining volume for all jobs at t in the algorithm is at most $\epsilon(|P|+|D|)+|F|$. Since the optimal solution is non-idle, this is also an upper bound for the total remaining volume in the optimal solution at t. However, all pending jobs in the optimal solution at t (with the exception of at most m jobs currently being processed) have a remaining processing time of at most $\frac{\mu}{2}$. Thus, the total number of pending jobs in the optimal solution at t is at most $m + \frac{2}{\mu} \cdot (\epsilon(|P|+|D|)+|F|)$, which is $O(m \cdot \mu^2 + \frac{|F|}{\mu})$. Those jobs are in P and D and of size at least $\frac{\mu}{2}$. Therefore the total weight of the optimal solution at time t is:

$$W^*(t) = O\left(m \cdot \mu + \frac{|F|}{\mu^2}\right).$$

We turn to analyze the weight of the algorithm at time t. Note that $|D| \leq \frac{m\cdot\mu^4}{2-2\frac{\epsilon}{\mu}} \leq \frac{m\cdot\mu^4}{1.5}$ since otherwise there must be a job $q \in D$ that $x_q < \mu - \epsilon$. The algorithm ALG, remains with the jobs in the sets P and F. Their combined size is $|P| + |F| = m \cdot \mu^4 - |D| \geq \frac{m\cdot\mu^4}{3}$. Since the algorithm ALG does not complete any of the jobs in F, it remains with weight of at least $|F|$ since the weight each job in F is 1. Therefore:

$$W^{ALG}(t) \geq |F|.$$

Analyzing in different way, the algorithm remains with at least $\frac{m\cdot\mu^4}{3}$ jobs (of the sets P and F) where the minimal weight of a job is $\frac{1}{\mu}$. Therefore, the weight of the algorithm ALG at time t is:

$$W^{ALG}(t) \geq \Omega\left(\frac{m\cdot\mu^4}{3} \cdot \frac{1}{\mu}\right) = \Omega(m \cdot \mu^3).$$

That means:

$$W^{ALG}(t) = \Omega(m \cdot \mu^3 + |F|).$$

Combining all, the local competitiveness at time t is:

$$\Omega\left(\frac{m \cdot \mu^3 + |F|}{m \cdot \mu + \frac{|F|}{\mu^2}}\right)$$

which is $\Omega(\mu^2)$ local competitive ratio. To finish the proof, we use the standard “bombardment” technique: at time t, we start releasing m jobs of size $\epsilon = \frac{1}{\mu}$ every ϵ time. The total weight of the released jobs in each interval is $m\mu$. This is easily seen to extend the $\Omega(\mu^2)$ lower bound from local competitiveness to general (global) competitiveness. ◀

5 Impossibility of online Immediate-Dispatching algorithms for Robust Scheduling

In this part, we will show lower bounds on immediate dispatching algorithms in the distortion-oblivious setting. We show that for both total flow time and total stretch objectives, there are polynomial lower bounds in terms of the number of jobs n for arbitrarily close to 1 values of μ and P, even if randomization is allowed. We first prove the lower bound version for deterministic algorithms (Theorem 12). Then we extend the proof to allow algorithms that use randomness (Theorem 13).

▶ **Theorem 12.** *For $m > 1$, every online immediate dispatching deterministic algorithm ALG, for every constant distortion factor μ, has a competitive ratio of $\Omega\big(\sqrt{\frac{n}{m}}\big)$ with respect to the total flow time objective.*

Proof. Theorem 12. At the beginning (i.e., time 0) we release k jobs of estimated size 1, the value of k will be determined later. Let x_i and be the number of jobs assigned to machine i by the algorithm ALG.

$$\sum_{i\in[m]} x_i = k.$$

Let j be the machine with the largest number of assigned jobs by the algorithm (i.e., $j = argmax_{i\in[m]}x_i$). For jobs assigned to machine j we set their actual size to be μ, and for the rest of the we set their actual size to 1.

The optimal solution, completes all the released jobs by time $t = \frac{x_j\mu+(m-1)x_j}{m}$, with *round robin* algorithm. The total flow time of the jobs until time t is $O(\frac{k^2}{m}\cdot\mu)$.

By that time, the algorithm remains with volume of at least $x_j\mu - t$ of remaining jobs assigned to machine j. Plug in the value of t and the remaining volume is:

$$x_j\mu - t = \Omega\bigg(\frac{k}{m}\cdot(\mu-1)\bigg).$$

From time t until time $t+(k/m)^2\cdot\mu$ we release in intervals of μ (i.e., $t, t+\mu, t+2\mu, t+3\mu, ...$) m jobs of actual processing time μ and estimated processing time 1. In total, during the construction we release $k + m\cdot(k/m)^2$ jobs.

At each interval, the optimal solution assigns the m released jobs to the m machines and completes them by the end of the interval. Thus the optimal solution total flow time is

$$F^* = m\cdot(k/m)^2\cdot\mu + O\bigg(\frac{k^2}{m}\cdot\mu\bigg) = O\bigg(\frac{k^2}{m}\cdot\mu\bigg).$$

The algorithm at time t remains with $\Omega\big(\frac{k}{m}(\mu-1)\big)$ volume (of jobs assigned to machine j), therefore, since the maximum processing time of a job is μ, at any tine between t and $t+(k/m)^2\cdot\mu$, the algorithm has at least $\Omega\Big(\frac{k}{m}(1-\frac{1}{\mu})\Big)$ alive jobs. Hence, the algorithm ALG total flow time is

$$F^{ALG} \geq (k/m)^2\cdot\mu\cdot\bigg(\frac{k}{m}\cdot\bigg(1-\frac{1}{\mu}\bigg)+m\bigg) + \Omega\bigg(\frac{k^2}{m}\cdot\mu\bigg).$$

Therefore the algorithm ALG is $\Omega(\frac{k}{m}(1-1/\mu))$-competitive with the optimal solution. The number of released jobs during execution is $n = k + m\cdot(k/m)^2 = O\Big(\frac{k^2}{m}\Big)$ and therefore, in other terms the algorithm ALG competitive ratio is $\Omega\big(\sqrt{\frac{n}{m}}(1-1/\mu)\big)$, which for a constant μ is $\Omega\big(\sqrt{\frac{n}{m}}\big)$. ◀

Now we prove the second theorem, Theorem 13 with the same technique we used at the proof of Theorem 12.

▶ **Theorem 13.** *For $m > 1$, every online immediate dispatching algorithm ALG that can use randomness, for every constant distortion factor μ, has a competitive ratio of $\Omega\Big(\frac{n^{1/4}}{\sqrt{m}}\Big)$ with respect to the total flow time objective.*

Proof. Theorem 13. We use Yao's principle and describe a lower bound for deterministic algorithms on a given distribution over the input. This yields a lower bound for randomized online algorithms.

Release k jobs of estimated size 1 at the beginning, $k/2$ of them with actual size 1 and the rest with actual size μ. The algorithm ALG assigns the jobs to the machines. Let x_i be the number of jobs that the algorithm assigned to machine i. Since the algorithm does not know the actual sizes at the assignment time, in expectation, every machine has $\frac{x_i}{2}$ jobs of actual size 1 and $\frac{x_i}{2}$ jobs of actual size μ.

We denote by z_i the number of jobs of actual size μ that are assigned to machine i by the algorithm. Similarly, we denote by y_i the number of jobs of actual size 1 that are assigned to machine i by the algorithm. Note that for every machine i, $z_i + y_i = x_i$.

The optimal solution, can perform *round robin* algorithm and complete all the released jobs at time $t = \frac{k(\mu+1)}{2m}$. The total flow time of the jobs until time t is $O\left(\frac{k^2}{m} \cdot \mu\right)$.

Let j be the machine with the largest number of assigned jobs by the algorithm (i.e., $j = argmax_{i\in[m]} x_i$). As before, $x_j \geq \frac{k}{m}$. The random variable z_j is a binomial random variable, $z_j \sim B(x_j, 0.5)$. Let G be the event that $z_j \geq \frac{k}{m} + \sqrt{\frac{k}{m}}$, note that the event G happens with probability at least $\frac{1}{4}$ (i.e., $P(G) \geq \frac{1}{4}$). We analyze only the case that G happens, in any other case we let the total flow time of the algorithm to be 0.

If G happens, then the algorithm ALG at time t is left with $\Omega\left(\sqrt{\frac{k}{m}} \cdot (\mu - 1)\right)$ of volume of jobs that assigned to machine j.

Repeating the last part of the previous proof, from time t until time $t + (k/m)^2 \cdot \mu$ we release in intervals of μ (i.e., $t, t+\mu, t+2\mu, t+3\mu, ...$) m jobs of actual processing time μ and estimated processing time 1. In total, during the construction we release $k + m \cdot (k/m)^2$ jobs.

At the beginning of each interval, the optimal solution assigns the m released jobs to the m machines and completes them at the end of each interval. Thus the optimal solution total flow time is

$$F^* = m \cdot (k/m)^2 \cdot \mu + O\left(\frac{k^2}{m} \cdot \mu\right) = O\left(\frac{k^2}{m} \cdot \mu\right).$$

At the time between t to $t + (k/m)^2 \cdot \mu$, ALG has at least $\frac{k}{m} \cdot (1 - 1/\mu) + m$ alive jobs. Hence, the algorithm ALG total flow time is

$$E[F^{ALG}] \geq P(G) \cdot \left((k/m)^2 \cdot \mu \cdot \left(\sqrt{\frac{k}{m}} \cdot (1 - 1/\mu) + m\right) + \Omega\left(\frac{k^2}{m} \cdot \mu\right)\right).$$

Therefore the algorithm ALG is $\Omega\left(\sqrt{\frac{k}{m}}(1 - 1/\mu)\right)$-competitive with the optimal solution. The number of released jobs during execution is $n = k + m \cdot (k/m)^2$ and therefore, the algorithm ALG competitive ratio is $\Omega(\frac{n^{1/4}}{\sqrt{m}}(1 - 1/\mu))$. ◀

The lower bound also applies for stretch since all the jobs are in the range of $[1, \mu]$, therefore considering the weights of the jobs can only decrease the competitive ratio by factor of μ, which is negligible related to the number of jobs n.

6 Discussion and Open Problems

In this paper, we present the first distortion-oblivious algorithms for total stretch, which have optimal competitive ratio. This provides an optimal deterministic distortion-obliviousness algorithm for the total stretch objective on both single and multiple machines. We also present nearly-optimal, distortion-oblivious algorithms for total flow time on multiple machines. We also show that with immediate dispatching no algorithm with sub-polynomial competitive ratio exists. It would be interesting to close the gap between the presented algorithms for flow time, that achieve $O(\mu \log P)$-competitiveness, to the previously known lower bound of $\Omega(\mu + \log P)$. Another direction of future work would be randomized distortion-oblivious algorithms for minimizing total stretch.

References

1 Algorithms with predictions. `https://algorithms-with-predictions.github.io`.

2 Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 11–18, New York, NY, USA, 2003. ACM. `doi:10.1145/777412.777415`.

3 Baruch Awerbuch, Yossi Azar, Stefano Leonardi, and Oded Regev. Minimizing the flow time without migration. *SIAM Journal on Computing*, 31(5):1370–1382, 2002. `doi:10.1137/S009753970037446X`.

4 Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1070–1080. ACM, 2021. `doi:10.1145/3406325.3451023`.

5 Yossi Azar, Stefano Leonardi, and Noam Touitou. Distortion-oblivious algorithms for minimizing flow time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 252–274. SIAM, 2022. `doi:10.1137/1.9781611977073.13`.

6 Eric Balkanski, Tingting Ou, Clifford Stein, and Hao-Ting Wei. Scheduling with speed predictions, 2022. `doi:10.48550/ARXIV.2205.01247`.

7 Nikhil Bansal, Kedar Dhamdhere, Jochen Könemann, and Amitabh Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. *Algorithmica*, 40(4):305–318, 2004. `doi:10.1007/s00453-004-1115-0`.

8 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.

9 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. `doi:10.1145/380752.380778`.

10 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.

11 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.

12 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, pages 285–294, New York, NY, USA, 2021. Association for Computing Machinery. `doi:10.1145/3409964.3461790`.

13 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.

14 Stefano Leonardi. *A Simpler Proof of Preemptive Total Flow Time Approximation on Parallel Machines*, pages 203–212. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. `doi:10.1007/11671541_7`.

15 Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 110–119, New York, NY, USA, 1997. ACM. doi:10.1145/258533.258562.

16 Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. *arXiv*, 2022. doi:10.48550/arXiv.2202.10199.

17 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. URL: http://proceedings.mlr.press/v80/lykouris18a.html.

18 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. URL: http://papers.nips.cc/paper/6782-revenue-optimization-with-approximate-bid-predictions.

19 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. doi:10.4230/LIPIcs.ITCS.2020.14.

20 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *arXiv preprint*, 2020. doi:10.48550/arXiv.2006.09123.

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

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

23 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. doi:10.1002/nav.3800030106.

A Total Flow Time Appendix

A.1 LCF algorithm analysis for total flow time

In this setting, we use the LCF (Lowest Class First) algorithm. The algorithm maintains a list of jobs L; first sorted by jobs' class and then by release time. At any time the algorithm processes the m first jobs of the list L.

▶ **Theorem 14.** *LCF is $O(\mu \log P)$ competitive for inputs with distortion μ.*

▶ **Theorem 15.** *LCF is $O(\mu \log \mu + \mu \log \frac{n}{m})$ competitive for inputs with distortion μ.*

▶ **Corollary 16.** *For every μ, LCF is $O(\min(\mu \log P, \mu \log \mu + \mu \log \frac{n}{m}))$-competitive for inputs with distortion μ.*

Proof of Theorem 14. We skip most of the proof since it is almost similar to the proof of Theorem 2. The proof is based on the following two lemmas, which their proof is omitted.

▶ **Lemma 17.** *If $t \in T$ then $\Delta_{\leq k} V(t) \leq m\mu_1 2^{k+1}$.*

▶ **Lemma 18.** *If $t \in T$ then $\delta_{\geq k_1 \leq k_2}(t)^{LCF} \leq m(\mu+3)(k_2 - k_1 + 2) + 2\delta^*_{\leq k_2}(t)$.*

Similar steps as in the proof of Theorem 2 and plugging in Lemma 18 concludes the proof. ◀

Proof of Theorem 15. Lets define the *group* of a job q as the unique integer i such that $p_q \in [2^i, 2^{i+1})$, differently from the class of the job, the *group* is defined over the actual processing time rather than the estimated.

Let $\bar{k}$ be the maximum integer k such that for some time $t \in T$, $\delta^{LCF}_{\geq k}(t) \geq m$ (note that $\delta^{LCF}_{\geq k}(t) \geq m$ is the number of alive jobs of *class* at least k, not *group*). If such integer does not exists, then the set T is empty. In addition, for this proof we define k_{min} differently; as the lowest *group* of a job. Let $T_j \subset T$, $j = k_{min}...\bar{k}$, be the set of times that all the machines are busy and the maximal currently processed job *group* is j. Let $T_{\bar{k}+1} \subset T$ be the set of times when all machines are busy and at least one machine is processing a job of *group* higher than $\bar{k}$. Observe that $T_{k_{min}}...T_{\bar{k}+1}$ defines a partition of T . We can write the total flow time of LCF as:

$$F^{LCF} = \int_{t\notin T} \delta^{LCF} dt + \int_{t\in T} \delta^{LCF} dt$$
$$= \int_{t\notin T} \gamma^{LCF}(t)dt + \sum_{j=k_{min}}^{\bar{k}+1} \int_{t\in T_j} \delta^{LCF}(t)dt.$$

Note that $\forall t \in T_j$ we have $\delta_{<j-\lceil\log\mu\rceil}(t) < m$ since the algorithm currently process a job of *group* j. Also note that for any time t we have $\delta_{>\bar{k}}(t) < m$ from the definition of $\bar{k}$. By these two observations we get that for any group j, $\forall t \in T_j$ $\delta^{LCF}(t) \leq 2m + \delta^{LCF}_{\geq j-\lceil\log\mu\rceil, \leq \bar{k}+1}(t)$.

$$F^{LCF} \leq \int_{t\notin T} \gamma^{LCF}(t)dt + \sum_{j=k_{min}}^{\bar{k}+1} \int_{t\in T_j} (2m + \delta^{LCF}_{\geq j-\lceil\log\mu\rceil, \leq \bar{k}+1}(t))dt.$$

By plugging in Lemma 18:

$$F^{LCF} \leq \int_{t\notin T} \gamma^{LCF}(t)dt + \sum_{j=k_{min}}^{\bar{k}+1} \int_{t\in T_j} (2m + m\mu(\bar{k} - j + 3 + \lceil\log\mu\rceil) + 2\delta^*_{\leq \bar{k}+1}(t))dt$$

which implies:

$$F^{LCF} \leq \int_{t\notin T} \gamma^{LCF}(t)dt + (3\mu + 2)\int_{t\in T} mdt + \sum_{j=k_{min}}^{\bar{k}+1} m\mu(\bar{k} - j)|T_j|$$
$$+ \sum_{j=k_{min}}^{\bar{k}+1} |T_j|m\mu\lceil\log\mu\rceil + 2\int_{t\in T} \delta^*_{\leq \bar{k}+1}(t)dt$$

where the first two terms $\int_{t\notin T} \gamma^{LCF}(t)dt + (3\mu + 2)\int_{t\in T} mdt$ are at most $(3\mu + 2)F^*$ since $\int_t \gamma^{ALG}(t) \leq F^*$ and for any $t \in T$ we have $\gamma^{ALG}(t) = m$. The fourth term, $\sum_{j=k_{min}}^{\bar{k}+1} |T_j|m\mu\lceil\log\mu\rceil$ equals $\mu\lceil\log\mu\rceil \int_{t\in T} \gamma^{ALG}(t)dt$ since T_j defines a partition of T. Using the fact $\int_{t\in T} \gamma^{ALG}(t)dt \leq F^*$ we have that the fourth term is bounded by $\mu\lceil\log\mu\rceil F^*$. The right most term, $\int_{t\in T} \delta^*_{\leq \bar{k}+1}(t)dt \leq F^*$, from the definition of local competitiveness. In addition, for $j = \bar{k}$ the mid term is negative. Therefore we can write:

$$F^{LCF} \leq (\mu\lceil\log\mu\rceil + 3\mu + 4)F^* + \mu \sum_{j=k_{min}}^{\bar{k}} m(\bar{k} - j)|T_j|.$$

Now we have the following lemma, which its proof is adapted from [14] (lemma 8). Their lemma was proved for SRPT. We show that it also holds for LCF. Note that in our proof, the sets T_j are defined according to the original *group* while in the original proof the sets T_j are defined according to the remaining processing time *group*.

▶ **Lemma 19.** $F(n) = \sum_{j=k_{min}}^{\bar{k}} m(\bar{k}-j)|T_j| = O(F^* \cdot \log \frac{n}{m})$.

Proof of Lemma 19. Let T_j^l be the set of times that machine l processes a job of *group* j. At every time $t \in T_j$ is also a part of m sets $T_{j_l}^l$ where j_l denotes the *group* of the job processed on machine ℓ. Also, at every time $t \in T_j$ the maximal group of a job is j, therefore $\bar{k} - j \leq \bar{k} - j_\ell$ for any machine ℓ and the following inequality follows:

$$F(n) = \sum_{j=k_{min}}^{\bar{k}} m(\bar{k}-j)|T_j| \leq \sum_{k_{min}}^{\bar{k}} \sum_{j \in [m]} (\bar{k}-j)|T_j^l|.$$

Every job of *group* i gives a contribution to the above equation of at most $(\bar{k}-i)2^{i+1}$. Let n_j denote the number of jobs of *group* j, therefore:

$$F(n) \leq \sum_{j=k_{min}}^{\bar{k}} (\bar{k}-j)n_j 2^{j+1}.$$

Note that this is only possible because we consider the *group* of the job, and therefore the maximal processing time of the job is 2^{j+1} rather than $2^{j+1} \cdot \mu_1$. Let $I_i = n_{\bar{k}-i} \cdot 2^{\bar{k}-i}$, for $i = k_{min}, ..., \bar{k}$. Then the sum becomes:

$$F(n) \leq 2 \cdot \sum_{i=0}^{\bar{k}-k_{min}} i \cdot I_i$$

In order to bound the function $F(n)$, we maximize the function subject to two obvious constraints:

$$\sum_{i=0}^{\bar{k}-k_{min}} I_i \leq \sum_q p_q$$
$$\sum_{i=0}^{\bar{k}-k_{min}} 2^i \cdot I_i \leq n \cdot 2^{\bar{k}}.$$

To complete the proof we use the following lemma proved in [14].

▶ **Lemma 20.** *Given a sequence $a_1, a_2, ...$ of non-negative numbers such that $\sum_{i \geq 1} a_i \leq A$ and $\sum_{i \geq 1} 2^i \cdot a_i \leq B$ then $\sum_{i \geq 1} i \cdot a_i \leq \log(\frac{4B}{A}) \cdot A$.*

We use Lemma 20 by setting $a_i = I_i$ for $i = 0, ..., \bar{k} - k_{min}$, $A = \sum_q p_q$ and $B = n \cdot 2^{\bar{k}}$.

$$\sum_{i=0}^{\bar{k}-k_{min}} i \cdot I_i \leq \log(\frac{4n \cdot 2^{\bar{k}}}{\sum_q p_q}) \cdot \sum_q p_q$$

By the definition of $\bar{k}$ and that $\sum_q p_q \leq F^*$ we get that:

$$F(n) \leq \sum_{i=0}^{\bar{k}-k_{min}} i \cdot I_i = O(F^* \cdot \log \frac{n}{m})$$

and that concludes the proof. ◀

Note that Lemma 19 concludes the proof, since:

$$F^{LCF} \leq (\mu \lceil \log \mu \rceil + 3\mu + 4) \cdot F^* + O(\mu \log \frac{n}{m}) \cdot F^* = O(\mu \log \mu + \mu \log \frac{n}{m}) \cdot F^*$$

as required. ◀

A.2 LCF and Algorithm 1 are $\Omega(\mu \log P)$-competitive

We argue that the theorems Theorem 14 and Theorem 2 are tight for $m \geq 2$. We show that by a single construction for both algorithms, that is built using $L = \lceil \log_\mu P \rceil - 1$ phases. We first describe the input construction, prove for the LCF algorithm and then explain that on this input both algorithms act exactly the same and the proof holds for both. For simplicity we assume that m and μ are even. Denote by r_i the time that phase i begins and is defined as follows:

$$P_i = \frac{P}{(2\mu)^i}.$$

$$r_0 = 0 \ ; \ r_i = r_{i-1} + 2P_i$$

For $i = 0, 1, ..., L-1$; at time r_i, we release $\frac{3m}{2}$ jobs of actual processing time P_i, their estimated processing time is also P_i (i.e., with no estimation error). Let $e_i = r_i + \frac{3P_i}{2}$. At time e_i we release $\frac{m\mu}{2}$ jobs of processing time $\frac{P_i}{\mu}$ with estimated time P_i.

The optimal solution can perform *round robin* algorithm and completes all the jobs that has been released at time r_i by time e_i and the jobs released at time e_i it completes by time r_{i+1}. Therefore at time r_L, the optimal solution completes all of the released jobs.

The algorithm LCF will always prefer jobs released at phase $i+1$ over phase i. Therefore, by time e_i LCF completes precisely m jobs that released at time r_i and the remaining $m/2$ jobs are processed precisely for $\frac{P_i}{2}$ time. In addition, LCF also process jobs of previous phases. *Inductively*, we show that for $i \geq 1$, those jobs are with remaining processing time of at least $2P_i$ at time r_i (and therefore also at $r_i + P_i$). Hence, the jobs from previous phases will be left with at least P_i processing time until phase $i+1$. Therefore neither of the remaining jobs at any phase until i is ever finished.

At the time between e_i and r_{i+1}, LCF continues to process the $m/2$ jobs released at time r_i and on the other $m/2$ machines it processes the jobs released by time e_i. LCF completes $\frac{m\mu}{4}$ and remains with $\frac{m\mu}{4}$ with processing time $\frac{P_i}{\mu}$ (unprocessed). Since $\frac{P_i}{\mu} \geq 2P_{i+1}$ this completes the induction.

At every phase, LCF added $\frac{m\mu}{4}$ jobs, that as discussed are not completed, and therefore

$$\delta^{LCF}(r_L) = \Omega(L \cdot m\mu) = \Omega(m\mu \log_\mu(P))$$

while the optimal solution remains with no jobs. To finish the proof, we use the standard "bombardment" technique: at time r_L, we start releasing m unit-jobs. This is easily seen to extend the $\Omega(\mu \log_\mu P)$ lower bound from local competitiveness to general (global) competitiveness.

Note that on this instance, since LCF does not need to do any migration and therefore Algorithm 1 actually behaves the same and the analysis also shows that Algorithm 1 analysis is tight.