Non-Preemptive Flow-Time Minimization via Rejections

Anupam Gupta
Carnegie Mellon University

Amit Kumar
IIT Delhi

Jason Li
Carnegie Mellon University

Abstract

We consider the online problem of minimizing weighted flow-time on unrelated machines. Although much is known about this problem in the resource-augmentation setting, these results assume that jobs can be preempted. We give the first constant-competitive algorithm for the non-preemptive setting in the rejection model. In this rejection model, we are allowed to reject an ε-fraction of the total weight of jobs, and compare the resulting flow-time to that of the offline optimum which is required to schedule all jobs. This is arguably the weakest assumption in which such a result is known for weighted flow-time on unrelated machines. While our algorithms are simple, we need a delicate argument to bound the flow-time. Indeed, we use the dual-fitting framework, with considerable more machinery to certify that the cost of our algorithm is within a constant of the optimum while only a small fraction of the jobs are rejected.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling, Rejection, Unrelated Machines, Non-Preemptive

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.70

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.09602`.

1 Introduction

Consider the problem of scheduling jobs for weighted flow-time minimization. Given a set of m unrelated machines, jobs arrive online and have to be processed on one of these machines. Each job j is released at some time r_j, has a potentially different processing requirement (size) p_{ij} on each machine i, and a weight w_j which is a measure of its importance. The objective function is the *weighted flow time* (or *response time*): if the job j completes its processing at time C_j, the flow/response time is $(C_j - r_j)$, i.e., the time the job spends in the system. The goal is now to minimize the weighted sum $\sum_j w_j(C_j - r_j)$.

The problem of flow-time minimization has been extensively studied both from theoretical and practical perspectives. The theoretical analyses have to assume that the jobs can be pre-empted in order to prove any meaningful competitive ratio, and it is easy to see why. If we schedule a long low-weight job and a large number of short high-weight items arrive meanwhile, we cannot afford to delay the latter (else we suffer large flow-time), so the only solution would be to preempt the former (See [14] for strong lower bounds.) And even with pre-emption, the problem turns out to be difficult for multiple machines: e.g., [11] show no bounded competitive ratio is possible for the case of unrelated machines. Hence, it is natural to consider models with "resource augmentation" where the algorithm has slightly more

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 70; pp. 70:1–70:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

resources than the adversary. E.g., in the speed-augmentation setting, where the algorithm uses machines of speed $(1+\varepsilon)$-times those of the adversary, Chadha et al. [7] showed how to get a preemptive schedule with weighted flow time at most poly$(1/\varepsilon)$ times the optimal flow time.

A different model of resource augmentation was proposed by Choudhury et al. [8] in the context of load balancing and maximum weighted flow-time, where we are allowed to reject at most ε-fraction of the total weight of the incoming jobs, but we compare with the optimum off-line algorithm which is required to process all the jobs. The motivation was two-fold: (a) the model is arguably more natural, since it does not involve comparing to an imaginary optimal schedule running on a slower machine, and (b) even with speed-augmentation, there are problems, e.g. on-line load balancing, where even a constant factor speed-up does not suffice to give meaningful results. Indeed, getting a non-preemptive schedule for weighted flow-time is one of these problems. Consider for example the following input: a job of unit size and unit weight at time 0 arrives. As soon as the algorithm schedules it, the adversary releases L jobs of size $\varepsilon \ll 1/L^2$. The optimal off-line flow-time is $O(1)$, but the algorithm will incur total flow-time of $\Omega(L)$. The model of job rejection is intuitively more powerful than speed-augmentation (although no such formal connection is known): loosely, the speed-augmentation model only allows us to uniformly reject an ε-fraction of each job, whereas the rejection model allows us to "non-uniformly" reject an arbitrary subset of jobs, as long as they contribute only an ε-fraction of the total weight.

1.1 Our Results

We consider the problem of non-preemptive scheduling on unrelated machines where the objective is to minimize total weighted flow-time of jobs. Our main result is the following:

▶ **Theorem 1** (Main Theorem). *For the problem of online weighted flow-time minimization on unrelated machines, there is a deterministic algorithm that rejects at most an ε-fraction of the total weight of incoming jobs, and ensures that the total weighted flow time for the remaining jobs is at most an $O(1/\varepsilon^3)$ factor times the optimal weighted flow time without rejections.*

Note that we compare with the off-line optimum which is allowed to be preemptive (in fact, migratory), but is required to process all the jobs. Our guarantees are, in fact, stronger. Define the notion of a "departure time" D_j for the job, which is the time at which either the job completes non-preemptively (in which case $D_j = C_j$) or is the time at which the job is rejected. A different natural definition of the total weighted response time in the presence of rejections would be the following:

$$\text{total weighted response time} := \sum_j w_j(D_j - r_j).$$

Keeping this quantity small forces us to decide on jobs early, and discourages us from letting jobs linger in the system for a long time, only to reject them at some late date. (Such a behaviour would be very undesirable for a scheduling policy, and would even be considered "unprofessional" in real-world settings.)

In fact the bulk of our work is in handling the single machine case. For this case, we get a slightly stronger bound.

▶ **Theorem 2** (Single Machine). *For the problem of online weighted flow-time minimization on a single machine, there is a deterministic algorithm that rejects at most an ε-fraction of the total weight of incoming jobs, and ensures that the total weighted flow time for the remaining*

jobs is at most $O(1/\varepsilon^2)$ factor times the optimal weighted flow time without rejections even when the offline optimum is given $(1+\varepsilon)$-extra speedup.

The fact that we can compare with an optimum offline algorithm which has faster machine allows us to use known immediate-dispatch algorithms for the setting of unrelated machines in a black-box manner [7, 2].

1.2 Our Techniques

Let us first focus on the single-machine case. Our algorithm rejects jobs in two different ways: some of the jobs are rejected immediately upon arrival, and others are rejected after receiving some processing. Moreover, assume for the moment that we are running a preemptive schedule, but without speed-augmentation. The high-level idea is to reject a "random" ε-fraction of jobs that come in. At an intuitive level, this rejects only ε-fraction of the weight (although this only in expectation, whereas we want this to hold deterministically at all times), and should create the effect of ε-speed augmentation. To implement this, let α_j be the "effect" of job j on the system – i.e., the increase in the total flow-time of the jobs currently in the system (assuming no future jobs arrive). The value of α_j also naturally corresponds to settings of dual variables for a natural flow-time LP. Using this we can (more-or-less) show that (a) the α_j values of the rejected jobs give us a lower bound on OPT, whereas (b) the α_j values of the non-rejected jobs upper-bound our cost. Hence, our goal becomes: at each time cancel *at most* an ε-fraction of the total weight $\sum_j w_j$, while cancelling *at least* an ε-fraction (say) of the total "dual" value $\sum_j \alpha_j$.

A little thought shows that this abstract task is hopeless in general for any deterministic strategy (say, if the α values rise very sharply), so we have to take the structure of the α_j values into account. We do this in two steps: we break the α_j contribution into α_j^+, the effect of job j on items denser than j, and α_j^-, its effect on less-dense items. Now we put jobs into buckets based on having the same (α^+, w) or (α^-, w) values, and rejecting each $1/\varepsilon^{th}$ job in each bucket. (The actual bucketing is a little finer, see §3.) Moreover, we reject the first job in each (α^+, w) bucket. The complications arise because we are more aggresive for each such (α^+, w) bucket, and because we may not have rejected any jobs in the (α^-, w) if it had less than $1/\varepsilon$ items. In §4.3.1 we perform a delicate charging to relate our aggressive rejections for the former to the total running time of the jobs, and show that (i) this aggressive rejection does not reject too much weight, and (b) also compensates for our timid rejections in the latter bucketing.

This high-level argument was done assuming preemptions. Since we want a non-preemptive schedule, only immediate rejections do not suffice, and we also must reject some jobs which we have started processing – indeed, if a large number of high-density ("important") jobs arrive right after we start processing some long low-density job j, delaying these more important jobs would cause large flow-time. So we must reject job j. However, as long as the total weight of these new jobs is w_j/ε, we can charge the rejection to these new jobs. This rejection makes the schedule very "unstable" and hence complicates the analysis. To get around this problem, we *mark* the job j as "preemptible". We then run a version of HDF with some preemptible and other non-preemptible jobs, and show that its performance can also be related to the LP variables.

Finally, for the multiple machines case we can perform a modular reduction to the single-machines case. We first use the *immediate dispatch* algorithm of Anand et al. [2] to assign jobs to machines, assuming speed augmenation. We then show our algorithm does well even compared to a stronger benchmark (i.e., where the offline schedule – instead of the online schedule – gets the speed augmentation). This gives us the theorem for the unrelated machines.

1.3 Related Work

There has been considerable work on the problem of minimizing total flow-time in the online setting, though most of it is in the preemptive setting. Several logarithmic competitive algorithms are known for unweighted flow-time on identical machines setting [15, 3], and in the related machines setting [10, 1], but there are strong lower bounds for the case of weighted flow-time even on a single machine [5]. In the restricted assignment settings with preemption, the unweighted flow-time problem becomes considerably harder even for 3 machines [7]. The situation for non-preemptive flow-time is much harder. Kellerer et al. [14] showed that one cannot achieve $o(n)$-competitive algorithm even for a single machine.

Much stronger results are known in the speed augmentation model, where machines in the online algorithm have ε-fraction more speed than the corresponding machines in the offline setting. This model was first proposed by Kalyanasundaram and Pruhs [13] for the problem of non-clairvoyant preemptive total flow-time minimization on a single machine. They gave an $O(1/\varepsilon)$-competitive algorithm for this problem. Chadha et al. [7] gave $O(1/\varepsilon^2)$-competitive preemptive algorithm for weighted flow-time in the unrelated machines setting. This was extended to the non-clairvoyant setting by Im et al. [12]. However, the non-preemptive weighted flow-time problem has strong lower bounds in the speed augmentation model even on a single machine [16].

The rejection model was proposed by Choudhury et al. [8] in the context of load balancing and maximum weighted flow-time in the restricted assignment setting. Lucarelli et al. [16] considered the non-preemptive scheduling problem of minimizing weighted flow-time in the unrelated machines setting. They showed that one can get $O(1/\varepsilon)$-competitive algorithm if we allow both $(1+\varepsilon)$-speed augmentation and rejection of jobs of total weight ε-times the total weight. Assuming both, we can design a much simpler algorithm and use the dual fitting techniques developed for speed augmentation models to give a simple analysis of this algorithm. Independently of us, Lucarelli et al. [17] recently announced an algorithm where they can remove the speed augmentation assumption for the simpler unweighted setting.

In the *prize-collection* model, one is allowed to incur a penalty term for the rejected jobs. This model has been widely studied, see e.g. Bartal et al. [6], Eppstein et al. [9], and Bansal et al. [4], though is considerably different from our model because here one can reject a large fraction of the jobs.

2 Definitions and Preliminaries

We consider the unrelated machine scheduling problem, as defined in §1. Our schedules will be non-preemptive. For a schedule $\mathcal{S}$, let $C_j^{\mathcal{S}}$ denote the completion time of j. We use $F_j^{\mathcal{S}}$ to denote the flow-time of j, and the objective function is given by $F^{\mathcal{S}} := \sum_j w_j \cdot F_j^{\mathcal{S}}$. We may remove the superscript $\mathcal{S}$ if it is clear from the context. We use $\mathcal{O}$ to denote the optimal off-line schedule. In Section 3, when considering the special case of a single machine, we use p_j to denote the *processing time* of job j (on this machine). Define the *density* ρ_j of a job as the ratio w_j/p_j. We assume that the parameter ε satisfies $\varepsilon^2 \leq 1/2$, and that $1/\varepsilon \in \mathbb{Z}$.

Fractional weighted flow-time. Given a schedule $\mathcal{A}$, let $p_j(t)$ denote the remaining processing time of job j at time t (assuming $t \geq r_j$). The remaining weight of j at time t is defined as $w_j(t) := \rho_j \cdot p_j(t)$. The weighted flow-time of j in this schedule is defined as $w_j(C_j - r_j)$, where C_j is the completion time of j. The fractional weighted flow-time of j is defined as $\sum_{t \geq r_j} w_j(t)$. Since $w_j(t) = 0$ for $t \notin [r_j, C_j]$, and $w_j(t) \leq w_j$ for any time t, it is clear that the fractional weighted flow-time is at most the (integral) weighted flow-time of j. The following claim is easy to check.

▶ **Claim 3.** If a job j is processed without interruption during $[t, t+p_j]$, then its fractional weighted flow-time is $w_j(t-r_j) + {}^{w_j p_j}/_2$. Moreover, if a job j gets rejected at time t', its weighted fractional flow-time is at least ${}^{w_j(t'-r_j)}/_2$.

Since the integral weighted flow-time of a job as in the claim above is $w_j(t-r_j)+w_j p_j$, we see the integer and fractional flow times are within factor of 2 of each other. Thus, for jobs which do not get preempted, we can argue about weighted fractional flow-time.

3 Algorithm for Single-Machine Weighted Flow Time

In this section, we consider the single-machine setting. For ease of algorithm description, we assume that all quantities are integers so that we can schedule jobs at the level of integer time-slots. We first describe an algorithm $\mathcal{A}$ which both rejects and preempts jobs. We subsequently show how to modify this algorithm (in an online manner) to another schedule which only rejects jobs, and does no preemptions. During our algorithm, we shall say that a job j is *active* at time t if it has been released by time t, but has not finished processing until time t, and has not been rejected. Let $A(t)$ denote the set of active jobs at time t in our algorithm. A subset of these jobs, denoted by $L(t)$, will be special – these jobs are allowed to be preempted (at time t). Once a job enters the set $L(t)$ at some time t, it stays in $L(t')$ for all subsequent times $t' \geq t$ until it finishes processing.

For a job $j \in A(t)$ and time t, recall that $p_j(t)$ denotes the remaining processing time. At every point of (integer) time t, the algorithm performs the following steps (in this order):

1. If job j arrives at time t, the algorithm may choose to reject it immediately upon arrival. We will call such rejections *immediate rejections*. If the job is not rejected, it gets added to the active set $A(t)$. For the moment, this is the only way in which a job gets rejected.
2. Let j be the job getting processed just before time t (i.e., in the time-slot $[t-1, t]$). If job j was not already in the set $L(t)$, the algorithm may move it to the set $L(t)$ if "many" jobs smaller than j have arrived during its execution. We will specify the precise rule soon. Recall that once added, the job j will remain in the set $L(t)$ until it finishes.
3. If the job j getting processed in the time-slot $[t-1, t]$ did not finish at time t and it is not in $L(t)$, the algorithm will continue to process j during the next time-slot $[t, t+1]$. Otherwise, if j finishes or $j \in L(t)$, the algorithm chooses a job in $A(t)$ which has the highest density (the HDF rule) and processes it during $[t, t+1]$.

Note that if multiple jobs arrive at a time t, we consider them in arbitrary order, and carry out the first two steps above iteratively for each such job, before executing step 3. This completes the description of the algorithm, except that we have not specified the rules for the first two steps.

We first explain the rule for adding a job to $L(t)$. Suppose the algorithm processes a job j during $[t-1, t]$, and suppose $j \notin L(t-1)$. Let t' be the time when the algorithm started processing j. Since it was not allowed to preempt j, it must have processed j without interruption during $[t', t]$. If the total weight of jobs arriving during $(t', t]$ exceeds w_j/ε, we add job j to the set $L(t)$. The intuition behind this rule is simple – the final algorithm will eventually reject all jobs which get added to the set $L(t)$, for all t. We can charge the weight of the rejected job j to the weight of the jobs which arrived during $[t', t]$. Moreover, consider a job j that does not get added to $L(t)$ over its lifetime. In a preemptive setting, we may have preempted such a job j on the arrival of a new shorter job, whereas here we perform such a preemption only when enough shorter jobs arrive. Since j was not added to $L(t)$, the total weight of such shorter jobs waiting on j is at most w_j/ε, so we can pay for the additional flow-time incurred by these shorter jobs (up to an $1/\varepsilon$ factor) by the flow-time of j.

The rule for immediate rejections is more involved. We maintain two tables T^+ and T^-. Each arriving job may get assigned to either T^+ or T^-, or both. We refer to each entry of these tables as a *bucket*. At a high level, every $(1/\varepsilon)^{th}$ job arriving in each bucket in either table suffers immediate rejection, though the details differ for the two tables. Let us elaborate on this further.

With every newly arriving job j, we specify a quantity α_j, which is the increase in the total flow-time of all the jobs in the system, assuming (i) no further jobs arrive after job j, and (ii) the scheduling algorithm follows the preemptive HDF policy from r_j onwards for *all* the jobs in $A(r_j)$. As in [2], we can write an expression for α_j as follows.

$$\alpha_j := \Big(w_j \sum_{j' \in A(r_j):\ \rho_{j'} \geq \rho_j} p_{j'}(r_j)\Big) \ + \ {}^{w_j p_j}/_2 \ + \ \Big(p_j \sum_{j' \in A(r_j):\ \rho_{j'} < \rho_j} w_{j'}(r_j)\Big). \tag{1}$$

We establish the convention that $A(r_j)$ does not contain job j. Moreover, if multiple jobs are released at time r_j, we consider them in arbitrary but fixed order, and add only those jobs to $A(r_j)$ which are considered before j.

For $x \in \mathbb{R}$, let $\llbracket x \rrbracket$ denote the largest integer i such that $2^i \leq x$. For a job j, define its *density-class* as $\llbracket \rho_j \rrbracket$. We partition jobs in $A(r_j)$ depending on their density-class as follows:

$$D_j^+ := \{j' \in A(r_j) \mid \llbracket \rho_{j'} \rrbracket \geq \llbracket \rho_j \rrbracket\} \qquad \text{and} \qquad D_j^- := \{j' \in A(r_j) \mid \llbracket \rho_{j'} \rrbracket < \llbracket \rho_j \rrbracket\}. \tag{2}$$

Now let α_j^+ be the terms in the expression for α_j involving jobs in D_j^+, and define α_j^- similarly. In other words,

$$\alpha_j^+ := \Big(w_j \sum_{j' \in D_j^+ :\ \rho_{j'} \geq \rho_j} p_{j'}(r_j)\Big) + \Big(p_j \sum_{j' \in D_j^+ :\ \rho_{j'} < \rho_j} w_{j'}(r_j)\Big); \tag{3}$$

$$\alpha_j^- := \Big(p_j \sum_{j' \in D_j^- :\ \rho_{j'} < \rho_j} w_{j'}(r_j)\Big). \tag{4}$$

Clearly, $\alpha_j = \alpha_j^+ + {}^{w_j p_j}/_2 + \alpha_j^-$. We now specify the definitions of the two tables.

- Table T^+: Buckets in this table are indexed by ordered pairs of integers (κ, λ). If an arriving job j satisfies $\alpha_j^+ \geq w_j p_j/\varepsilon$, we assign it to the bucket indexed $(\llbracket {}^{\alpha_j^+}/_{w_j} \rrbracket, \llbracket w_j \rrbracket)$ in this table, and add it to the set J^+ of jobs assigned to T^+. For each bucket, we cancel the first job that is assigned to that bucket, and then every $(1/\varepsilon)^{th}$ subsequent job assigned to it.
- Table T^-: Buckets in this table are indexed by ordered triplets of integers (γ, δ, η). Each arriving job which satisfies $\alpha_j^- > w_j p_j/\varepsilon$ is assigned to the bucket indexed $(\llbracket \alpha_j^- \rrbracket, \llbracket \rho_j \rrbracket, \llbracket p_j \rrbracket)$, and added to the set J^- of jobs assigned to T^-. For each bucket, cancel every $(1/\varepsilon)^{th}$ job assigned to this bucket. Note the subtle difference with respect to T^+: here the first job to be canceled in a bucket is the $(1/\varepsilon)^{th}$ job assigned to it.

3.1 The Final Algorithm $\mathcal{B}$

The actual online algorithm $\mathcal{B}$ is almost the same as $\mathcal{A}$, except when the algorithm $\mathcal{A}$ processes a job in $L(t)$ during time-slot $[t, t+1]$, the algorithm $\mathcal{B}$ idles, leaving this slot empty. In other words, when a job being executed is added to $L(t)$, the algorithm $\mathcal{B}$ rejects the job instead of eventually finishing it, perhaps after some preemptions. (We can think of this as being a *delayed rejection*, as opposed to the *immediate rejection* that $\mathcal{A}$ performs based on the above bucketing strategy.) Clearly, we can implement $\mathcal{B}$ in an online manner.

4 Analyzing the Single-Machine Algorithm

In this section, we provide the analysis of our single-machine algorithm $\mathcal{B}$. Naturally, the two main steps are to show that (i) an $O(\varepsilon)$ fraction of jobs by weight get rejected, and (ii) the total flow time is competitive with the optimal offline algorithm.

Showing (i) is relatively straightforward: a rejected job is either immediately rejected or is later rejected in $\mathcal{B}$ due to its preemption in $\mathcal{A}$. We will show that the rejected jobs falling under each of the two categories is an $O(\varepsilon)$ fraction by weight, with a separate analysis for each category. Both of the analyses are in Section 4.1.

To show flow time competitiveness of algorithm $\mathcal{B}$, we instead focus on bounding the tota flow time of algorithm $\mathcal{A}$. By Claim 3, the total (integer) flow-time of jobs that $\mathcal{B}$ does not reject is within a factor of two of their fractional flow-time in $\mathcal{A}$, since these are precisely the jobs that $\mathcal{A}$ does not preempt. Therefore, to prove Theorem 2, it suffices to show that $\mathcal{A}$ is $O(1/\varepsilon^2)$ factor competitive with the optimal offline algorithm.

Let $J^{\texttt{immed}}$ denote the set of jobs which get rejected immediately upon arrival, and let $\mathcal{O}$ denote the optimal offline schedule and $F^{\mathcal{O}}$ its fractional weighted flow time. Roughly speaking, our goal is to establish the following chain of approximate inequalities:

$$\varepsilon F^{\mathcal{A}} \lesssim \varepsilon \sum_j \alpha_j \lesssim \sum_{j \in J^{\texttt{immed}}} \alpha_j \lesssim F^{\mathcal{O}}, \tag{5}$$

where $\lesssim$ hides additive $\sum_j {}^{w_j p_j}/_{\varepsilon^{O(1)}}$ factors. Since $F^{\mathcal{O}} \geq \sum_j w_j p_j/2$, these additive losses still provide a ${}^1/_{\varepsilon^{O(1)}}$ competitive ratio.

For the first inequality, we will bound the flow time of algorithm $\mathcal{A}$, modulo an additive $\sum_j {}^{w_j p_j}/_{\varepsilon}$ factor, by the sum of α_j over all jobs $j \notin J^{\texttt{immed}}$, which are precisely the jobs that are finished by $\mathcal{A}$. We do so by exploiting the facts that the α_j values indicate an increase in flow time to an HDF algorithm, and that $\mathcal{A}$ is "approximately" an HDF algorithm. The details are in Lemma 5.

The second inequality is the most technically involved section of the paper. Not only does the immediate rejection scheme reject an $O(\varepsilon)$ fraction of jobs, but it also rejects jobs constituting an ε fraction of the total α_j value. The analysis is in Section 4.3.

Finally, the last inequality relates the optimal offline flow time to the sum of the α_j values of immediately rejected jobs. It is restated as Lemma 6 and proved in the appendix.

4.1 Bounding Weight of Rejected Jobs

In this section, we show that the total weight of rejected jobs is only an $O(\varepsilon)$ fraction of total. Recall that jobs either suffer immediate rejection, or are added to $L(t)$ for some time t, and hence suffer delayed rejection.

Let us first bound the total weight of the set $L := \cup_t L(t)$. For a job j in $L(t)$, let s_j be the first time when it gets processed and l_j be the time at which it enters the set $L(t)$. Since j must be processed uninterrupted in this interval $(s_j, l_j]$, the intervals associated with different jobs are disjoint. Moreover job j entered $L(t)$ because the total weight of jobs released during $(s_j, l_j]$ is at least w_j/ε. Thus the total weight of jobs in L can be upper bounded by ε times the weight of all the jobs.

We now account for the weight of jobs which are rejected immediately on arrival. For job j, let $[\![w_j]\!]$ denote the *weight-class* of this job. Jobs assigned to a bucket in T^+ have the same weight-class, by construction of the buckets. Jobs assigned to a bucket in T^- have the same $[\![\rho_j]\!]$ and $[\![p_j]\!]$, which pins down their weight $w_j = \rho_j \cdot p_j$ up to a factor of 4. This gives us the following facts:

- Since we reject every $(1/\varepsilon)^{th}$ job in each bucket of T^-, the total weight of jobs in J^- which get rejected immediately is at most 4ε times the weight of all jobs in J^-.
- Let J_f^+ be the subset of jobs in J^+ which happen to be the first jobs to be assigned to their respective buckets in T^+. Then the weight of all jobs in $J^+ \setminus J_f^+$ which get rejected immediately on arrival is at most 2ε times the total weight of all the jobs in J^+.

So it remains to account for the items items in J_f^+, which are all rejected. Recall that a job in J^+ is assigned to the bucket indexed $(\llbracket \alpha_j^+/w_j \rrbracket, \llbracket w_j \rrbracket)$ in T^+. Jobs in J_f^+ are assigned to distinct buckets in T^+. Fix an integer γ, and let J^γ denote the jobs in J_f^+ which are mapped to a bucket indexed (γ, κ) for some κ. The jobs in J^γ have distinct weight-classes and so it suffices to bound the weight of the highest weight job in J^γ – let this heaviest job be j^γ. Let S denote the set of such jobs j^γ as we range over all γ. Jobs in S have distinct $\llbracket \alpha_j^+/w_j \rrbracket$ values. Let $\Gamma = \{\gamma_1 < \gamma_2 < \ldots < \gamma_k\}$ be the integers γ for which there is a job $j^\gamma \in S$, and let the corresponding jobs in S be called $j_1, j_2, \ldots, j_k$.

Now starting from the smallest index in Γ, we charge each job $j_r \in S$ to a subset of jobs of total weight at least w_{j_r}/ε. The job j_r may charge to a job fractionally – if it charges to a fraction δ of some job j, then it can only use δw_j amount of weight of j for its charging (and we say that "j_r *charges to* δp_j *size of this job* j"). Of course, we need to ensure that the total fraction charged to a job is at most 1. We inductively maintain the following invariant for all $r \in 1 \ldots k$:

- The job j_r charges to jobs of total (fractional) weight at least $w_{j_r}/8\varepsilon$.
- Jobs $j_1, \ldots, j_r$ charge to jobs of total (fractional) size at most 2^{γ_r}.

Assuming these invariants hold for $r-1$, we show that they hold for r as well. Let $\rho^\star := \llbracket \rho_{j_r} \rrbracket$ be the density class for job j_r. By j_r's choice of bucket, $\llbracket \alpha_{j_r}/w_{j_r} \rrbracket = \gamma_r$, so

$$\alpha_{j_r}^+ \geq 2^{\gamma_r} \cdot w_{j_r}. \tag{6}$$

Recall from (2) that $D_{j_r}^+$ is the set of jobs of density class $\rho^\star$ or higher which are active at the time j_r is released. Let $P_r := \sum_{j \in D_{j_r}^+} p_j$ be the total processing time of these jobs. By (3), it follows that

$$\alpha_{j_r}^+ \leq w_{j_r} P_r. \tag{7}$$

Combining (6) and (7), $P_r \geq 2^{\gamma_r}$. By the second invariant, the first r jobs $j_1, \ldots, j_{r-1}$ have only charged to jobs of total size at most $2^{\gamma_{r-1}}$, so we can find jobs in $D_{j_r}^+$ of total (fractional) size $2^{\gamma_r} - 2^{\gamma_{r-1}} \geq 2^{\gamma_r - 1}$ which have not been charged yet, and charge to them. This proves the second invariant.

To prove the first invariant, we know that $\alpha_{j_r}^+ \geq w_{j_r} p_{j_r}/\varepsilon$, else j_r would not be assigned to T^+. Moreover, $\alpha_{j_r}^+ \leq w_{j_r} 2^{\gamma_r + 1}$ by the bucketing, so $2^{\gamma_r} \geq p_{j_r}/2\varepsilon$. Consequently, we charge to jobs of total size at least $2^{\gamma_r - 1} \geq p_{j_r}/4\varepsilon$, and these jobs have density class at least $\rho^\star$. Since $2\rho^\star \geq \rho_{j_r} = w_{j_r}/p_{j_r}$, we get their total (fractional) weight is at least $w_{j_r}/8\varepsilon$. This proves the first invariant, and hence the following theorem.

▶ **Theorem 4** (Few Rejections). *The weight of jobs suffering immediate rejection, plus those in $\cup_t L(t)$, is at most an $O(\varepsilon)$ fraction of the weight of all jobs released.*

4.2 Bounding the Weighted Fractional Flow-time

Next we show that the total fractional flow-time of $\mathcal{A}$ can be bounded in terms of total α_j values. We first focus on relating $F^{\mathcal{A}}$ to the sum of the α_j values, as described in (5).

Observe that α_j denotes the increase in objective function due to the arrival of j if we had followed the preemptive HDF policy for all the jobs from time r_j onwards. However, we follow a slightly different policy – if j' denotes the job that was running on the machine at time j's release time r_j, we let j' run until it finishes, or else until j' belongs to the set $L(t')$ at some time $t' \geq r_j$. If no further jobs are released after j, the HDF policy after this time t' would be non-preemptive. Thus, we would still expect that the total fractional weighted flow-time of our algorithm to be close to $\sum_j \alpha_j$. We formalise this intuition now. For every job j, we define a job $\phi(j)$ as follows: let j' be the job which was running just before time r_j (i.e., in the slot $[r_j - 1, r_j]$). If $j' \notin L(r_j)$, we define $\phi(j)$ to be j', otherwise we leave $\phi(j)$ undefined. Our policy for adding a job to the set $L(t)$ ensures that for every job j, $w(\phi^{-1}(j))$ is at most w_j/ε. [1] Recall that $J^{\texttt{immed}}$ is the set of jobs which get rejected immediately upon arrival. The following lemma, whose proof is deferred to the full version, states that the fractional weighted flow-time of the algorithm can be charged to the α_j values of the jobs which get immediately rejected.

▶ **Lemma 5.** *The fractional weighted flow-time of $\mathcal{A}$ is at most $\sum_{j:j\notin J^{\texttt{immed}}} \alpha_j + \sum_j w_j p_j/\varepsilon$.*

Proof. Jobs in $J^{\texttt{immed}}$ get rejected immediately, so their flow-time is 0. We now consider the jobs which are not immediately rejected in the rest of the proof. Consider the jobs in order of increasing release times. Let Δ_j denote the increase in the objective function value due to arrival of j. In other words, if J_1 is the set of jobs released before j, then Δ_j equals the total fractional weighted flow-time of $\mathcal{A}$ on the input $J_2 := J_1 \cup \{j\}$ minus that on the input J_1. The total weighted flow time of $\mathcal{A}$ on the entire input would be $\sum_j \Delta_j$, the sum of these increases. We now show that

$$\Delta_j \leq \alpha_j + w_j p_{\phi(j)}. \tag{8}$$

Since $w(\phi^{-1}(j')) \leq w_{j'}/\varepsilon$, we get that $\sum_j w_j p_{\phi(j)} = \sum_{j'} w(\phi^{-1}(j'))p_{j'} \leq \sum_{j'} w_{j'} p_{j'}/\varepsilon$. Hence, summing (8) over all j which are not in $J^{\texttt{immed}}$ proves the lemma.

Now we prove (8). Since we will be dealing with two inputs, J_1 and J_2, we parameterise all quantities by J_1 or J_2 to clarify which input we refer to. For example, $A(J_k, t), k = 1, 2$ will refer to the active set $A(t)$ on input J_k. Let $F(J_k, t)$ denote the fractional weighted flow-time of jobs in $A(J_k, t)$ *beyond* time t, i.e., $F(J_k, t) := \sum_{t' \geq t} \sum_{j \in A(J_k,t')} w_j(t')$.

There are two cases when job j arrives. If $\phi(j)$ is undefined, the job j' running in slot $[r_j - 1, r_j]$ belongs to $L(r_j)$. Hence the algorithm $\mathcal{A}$ on both inputs J_1, J_2 just runs HDF starting at time r_j. The difference between the corresponding flow times is precisely α_j, by definition.

Otherwise $\phi(j)$ is well-defined. Since j is the latest arrival, the job $\phi(j)$ will not be preempted, and runs to completion. Say job $\phi(j)$ completes at time t'. During the time $[r_j, t']$ the difference in fractional weighted flow-time between the two runs is precisely $w_j \cdot (t' - r_j) \leq w_j p_{\phi(j)}$. After time t' we run HDF on the remaining jobs, and the difference in the fractional weighted flow-time of the two runs is precisely what α_j would have been had j arrived at time t' instead of time r_j. In other words, if $J' := A(J_k, r_j) \setminus \{\phi(j)\}$,

$$\begin{aligned} F(J_2, t') - F(J_1, t') &= w_j p_j/2 + \sum_{j' \in J' : \rho_{j'} \geq \rho_j} w_j p_{j'}(t') + \sum_{j' \in J' : \rho_{j'} < \rho_j} w_{j'}(t') p_j \\ &= w_j p_j/2 + \sum_{j' \in J' : \rho_{j'} \geq \rho_j} w_j p_{j'}(r_j) + \sum_{j' \in J' : \rho_{j'} < \rho_j} w_{j'}(r_j) p_j \end{aligned}$$

[1] For a set S of jobs, let $w(S)$ denote the total weight of jobs in S.

But this is a subset of the terms of α_j: indeed, we're just missing the term corresponding to job $\phi(j)$. Hence, the total difference is at most $\alpha_j + w_j p_{\phi(j)}$, proving (8). ◀

To bound our flow time against the optimum using this lemma, note that $\sum_j w_j p_j/\varepsilon \leq 2F^{\mathcal{O}}/\varepsilon$, where we recall that $\mathcal{O}$ denotes the optimal offline schedule, and $F^{\mathcal{O}}$ its fractional weighted flow time. So we just need to bound $\sum_j \alpha_j = \sum_j {}^{w_j p_j}/_2 + \sum_j \alpha_j^+ + \sum_j \alpha_j^-$. The first term is again bounded by $F^{\mathcal{O}}$, so the work is in bounding the other two terms. We first record a convenient lemma – its proof is based on LP duality arguments and construction of dual variables are similar to those in [2], and is deferred to the full version.

▶ **Lemma 6** (Duality-based Lower Bound on OPT). $\sum_{j \in J^{\texttt{immed}}} \alpha_j \leq F^{\mathcal{O}} + \sum_j {}^{w_j p_j}/_{\varepsilon}$.

4.3 Controlling the α Terms

In this section, our goal is to establish the approximate inequality $\varepsilon \sum_j \alpha_j \lesssim \sum_{j \in J^{\texttt{immed}}} \alpha_j$, introduced in (5).

▶ **Lemma 7.** $\sum_j \alpha_j^+ \leq O(1/\varepsilon) \cdot \big(\sum_j w_j p_j + \sum_{j \in J^{\texttt{immed}}} \alpha_j^+\big)$.

Proof. The definition of J^+ implies that $\sum_{j \notin J^+} \alpha_j^+ \leq \sum_{j \notin J^+} w_j p_j/\varepsilon$. It remains to bound $\sum_{j \in J^+} \alpha_j^+$. We do an accounting per bucket in T^+. Fix a bucket B indexed by a pair (κ, λ), i.e., all jobs j in this bucket have $\llbracket \alpha_j^+/w_j \rrbracket = \kappa$, and $\llbracket w_j \rrbracket = \lambda$. Hence, if j is any job in this bucket, then $2^\kappa \leq \alpha_j^+/w_j \leq 2^{\kappa+1}$, and $2^\lambda \leq w_j \leq 2^{\lambda+1}$. Multiplying, $2^{\kappa+\lambda} \leq \alpha_j^+ \leq 4 \cdot 2^{\kappa+\lambda}$, i.e., the α_j^+ values of any two jobs in this bucket differ by a factor of at most 4.

Let J_B denote the jobs in J^+ assigned to this bucket B, and n_B denote their cardinality $|J_B|$. Since we reject the first job and then every subsequent $(1/\varepsilon)^{th}$ job in J_B, we immediately reject at least $\varepsilon\, n_B$ jobs in J_B. Therefore,

$$\sum_{j \in J_B} \alpha_j^+ \leq \frac{4}{\varepsilon} \cdot \sum_{j \in J_B \cap J^{\texttt{immed}}} \alpha_j^+.$$

Summing over all buckets, the lemma follows. ◀

▶ **Lemma 8.** $\sum_j \alpha_j^- \leq O(1/\varepsilon) \cdot \big(\sum_j w_j p_j + \sum_{j \in J^{\texttt{immed}}} \alpha_j\big)$.

Proof. The argument is similar to Lemma 7 in spirit, but technically more involved. The reason is that we do not remove any jobs from a bucket of T^- until it has $1/\varepsilon$ jobs assigned to it. Hence, for a bucket B, if J_B is non-empty but $|J_B| \leq 1/\varepsilon$, we have $J_B \cap J^{\texttt{immed}} = \varnothing$. However, if J_f^- is the set of jobs in J^- which are the first jobs assigned to their corresponding buckets in T^-, then we get (as in the proof of Lemma 7) that

$$\sum_j \alpha_j^- \leq O(1/\varepsilon) \cdot \left(\sum_j w_j p_j + \sum_{j \in J^{\texttt{immed}}} \alpha_j^- + \sum_{j \in J_f^-} \alpha_j^- \right). \tag{9}$$

It remains to bound $\sum_{j \in J_f^-} \alpha_j^-$, which we accomplish via the following claim. Since the proof is more technical, we defer it to the next section.

▶ **Claim 9.** $\sum_{j \in J_f^-} \alpha_j^- \leq O(\varepsilon) \cdot \big(\sum_j w_j p_j + \sum_j \alpha_j^+\big)$.

Combining this with (9) and Lemma 7, using that $\alpha_j^+ + {}^{w_j p_j}/_2 + \alpha_j^- = \alpha_j$, the lemma follows. ◀

Combining Lemmas 7 and 8 along with Lemma 5 and Lemma 6, we get

▶ **Theorem 10.** *The fractional weighted flow-time of the non-rejected jobs in $\mathcal{A}$ is $O(F^{\mathcal{O}}/\varepsilon^2)$.*

4.3.1 Proof of Claim 9

Define $\Lambda^+ := \sum_j \alpha_j^+$. Recall that for a job j, its density class is given by $\lfloor \rho_j \rfloor = \lfloor w_j/p_j \rfloor$. For each density class $\delta \in \mathbb{Z}$, let us define some notation:

- Let $A^\delta(t) := \{j \in A(t) \mid \lfloor \rho_j \rfloor = \delta\}$ denote jobs in $A(t)$ whose density class is δ.
- Let $P^\delta(t) := \sum_{j \in A^\delta(t)} p_j(t)$ and $W^\delta(t) := \sum_{j \in A^\delta(t)} w_j(t)$ be the total processing time and residual weight of jobs in $A^\delta(t)$, respectively. Since all jobs in this set have the same density class, observe that $\frac{W^\delta(t)}{P^\delta(t)}$ also lies in the range $[2^\delta, 2^{\delta+1})$.
- Define $P^\delta := \max_t P^\delta(t)$ and $W^\delta := \max_t W^\delta(t)$.

Our proof shows that $\sum_\delta P^\delta W^\delta$ is small; then we bound $\sum_{j \in J_f^-} \alpha_j^-$ by $\sum_\delta P^\delta W^\delta$. The proof of the following technical lemma is deferred to the full version.

▶ **Lemma 11.** $\sum_\delta P^\delta W^\delta \le O(1) \cdot \left(\sum_j w_j p_j + \Lambda^+ \right)$.

▶ **Lemma 12.** $\sum_{j \in J_f^-} \alpha_j^- \le O(\varepsilon) \cdot \sum_\delta P^\delta W^\delta$.

Proof. Let us first give a general method for bounding α_j^- of any job $j \in J^-$, and then we can apply it to the jobs in $J_f^- \subseteq J^-$. Recall that the jobs which contribute to α_j^- are the ones with a strictly smaller density class than that of j. We now show that one need not look at jobs of all such classes, and a subset of these classes suffice. Fix a job $j \in J^-$ of density class δ, and define an index set $\mathbb{I}_j$ as $\{\theta < \delta \mid P^\theta(r_j) \ge (1.5)^{\delta-\theta} p_j/8\varepsilon\}$.

▶ **Claim 13.** For any job $j \in J^-$ with density class δ, $\alpha_j^- \le 4p_j \cdot \sum_{\theta \in \mathbb{I}_j} W^\theta$.

Proof. Let j' be a job in $A(r_j)$ of strictly lower density class than j. Its contribution towards α_j^- is $p_j w_{j'}(r_j)$. Therefore, α_j^- is at most

$$\sum_{\theta<\delta} p_j W^\theta(r_j) = p_j \cdot \sum_{\theta \in \mathbb{I}_j} W^\theta(r_j) + p_j \cdot \sum_{\theta \notin \mathbb{I}_j, \theta<\delta} W^\theta(r_j). \tag{10}$$

Let us bound the summation from the second expression.

$$\sum_{\theta \notin \mathbb{I}_j, \theta<\delta} W^\theta(r_j) \le \sum_{\theta \notin \mathbb{I}_j, \theta<\delta} 2^{\theta+1} P^\theta(r_j) \le \sum_{\theta<\delta} \frac{(1.5)^{\delta-\theta}}{2^{\delta-\theta}} \cdot \frac{2^\delta p_j}{4\varepsilon} \le \frac{3w_j}{4\varepsilon}. \tag{11}$$

Substituting (11) into (10), and using that $\alpha_j^- \ge w_j p_j/\varepsilon$ for all jobs $j \in J^-$, we get that $\alpha_j/4 \le p_j \sum_{\theta \in \mathbb{I}_j} W^\theta(r_j) \le p_j \sum_{\theta \in \mathbb{I}_j} W^\theta$, which proves the desired result. ◀

Recall that job $j \in J^-$ is mapped in table T^- to the bucket indexed by $(\lfloor \alpha_j^- \rfloor, \lfloor \rho_j \rfloor, \lfloor p_j \rfloor)$. For a fixed pair (δ, η), consider the jobs in J_f^- which are mapped to buckets indexed (γ, δ, η) with various values of γ, and denote these jobs by $J_{(\delta,\eta)}$. Since J_f^- only contains the first job in each bucket, the $\lfloor \alpha_j^- \rfloor$ values of the various jobs in $J_{(\delta,\eta)}$ are all distinct. It follows that if $j^\star$ is the job in $J_{(\delta,\eta)}$ with the highest α_j^- value, then $\sum_{j \in J_{(\delta,\eta)}} \alpha_j^- \le 4\alpha_{j^\star}^-$. Thus, we just need to worry about one job per $J_{(\delta,\eta)}$ – let S denote this set of jobs.

The ordered pairs $(\lfloor \rho_j \rfloor, \lfloor p_j \rfloor)$ corresponding to jobs $j \in S$ are all distinct. For density class δ, let S^δ denote the jobs in S with density class δ. Using Claim 13,

$$\sum_{j \in S^\delta} \alpha_j^- \le 4 \sum_{j \in S^\delta} p_j \sum_{\theta \in \mathbb{I}_j} W^\theta = 4 \sum_{\theta<\delta} W^\theta \sum_{j \in S^\delta : \theta \in \mathbb{I}_j} p_j. \tag{12}$$

The jobs in S^δ also have different $[\![p_j]\!]$ values, so the sum $\sum_{j\in S^\delta:\theta\in\mathbb{I}_j} p_j \le 4p_{j'}$ for the job $j' := \arg\max\{p_j \mid j \in S^\delta, \theta \in \mathbb{I}_j\}$. By definition of $\mathbb{I}_j$, $p_{j'} \le 8\varepsilon P^\theta/(1.5)^{\delta-\theta}$. Substituting into (12),

$$\sum_{j\in S^\delta} \alpha_j^- \le 16 \sum_{\theta<\delta} \frac{8\varepsilon\, W^\theta P^\theta}{(1.5)^{\delta-\theta}}. \tag{13}$$

To complete the argument, $\sum_{j\in J_f^-} \alpha_j^-$ is at most

$$4\sum_\delta \sum_{j\in S^\delta} \alpha_j^- \overset{\text{eq.(13)}}{\le} 2^9\varepsilon \sum_\delta \sum_{\theta<\delta} \frac{W^\theta P^\theta}{(1.5)^{\delta-\theta}} = 2^9\varepsilon \sum_\theta W^\theta P^\theta \cdot \sum_{\delta>\theta} \frac{1}{(1.5)^{\delta-\theta}} = O\Big(\varepsilon \sum_\theta W^\theta P^\theta\Big).$$

This completes the proof of Lemma 12. ◀

Combining Lemmas 11 and 12 completes the proof of Claim 9, and hence for Theorem 10. In the full version of the paper we show that the algorithm is competitive even against an optimal algorithm that is allowed $(1+\varepsilon)$-speed augmentation – and hence prove Theorem 2.

5 Extension to Unrelated Machines

The extension of our result on single machine to the more general scenario of unrelated machines can be done very modularly. We shall use the following result from [7, 2].

▶ **Theorem 14.** *There is an online algorithm $\mathcal{D}$ which dispatches each arriving job j immediately upon arrival to one of the m machines such that the following property holds: if $J^{(i)}$ is the set of jobs which are dispatched to machine i and $\mathcal{O}^{\varepsilon',i}$ is the optimal solution to $J^{(i)}$ when we have only one machine with speed $(1+\varepsilon')$, then $\sum_i F^{\mathcal{O}^{\varepsilon',i}}$ is at most $1/\varepsilon'$ times the optimal weighted flow-time of J.*

The algorithms in [7, 2] actually build a schedule as well and use this schedule to immediately dispatch a job. The algorithm $\mathcal{D}$ can build this schedule in the *background* and use it to dispatch jobs, but not use it for actual processing. It follows from Theorem 14 and our result showing that our algorithm is also competitive against an optimal algorithm that is allowed $(1+\varepsilon)$-speed augmentation (which we defer to the full version), that if we run our algorithm on each of the machines i (with input $J^{(i)}$ arriving on-line) independently, then the total weighted flow-time of non-rejected jobs in our algorithm is at most $O(1/\varepsilon^3)$ times the optimal value.

References

1 S Anand. *Algorithms for flow time scheduling.* PhD thesis, Indian Institute of Technology, Delhi, 2013.

2 S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA'12*, pages 1228–1241. ACM, New York, 2012.

3 Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA*, pages 11–18, 2003.

4 Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Kedar Dhamdhere. Scheduling for flow-time with admission control. In *Proc. ESA, 2003*, pages 43–54. Springer, 2003. doi:10.1007/978-3-540-39658-1_7.

5 Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *SODA*, pages 1238–1244, 2009.

6 Yair Bartal, Stefano Leonardi, Alberto Marchetti-Spaccamela, Jiri Sgall, and Leen Stougie. Multiprocessor scheduling with rejection. *SIAM J. Discrete Math.*, 13(1):64–78, 2000.

7 Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In *STOC'09*, pages 679–683. ACM, New York, 2009.

8 Anamitra Roy Choudhury, Syamantak Das, Naveen Garg, and Amit Kumar. Rejecting jobs to minimize load and maximum flow-time. *J. Comput. System Sci.*, 91:42–68, 2018.

9 Leah Epstein and Hanan Zebedat-Haider. Preemptive online scheduling with rejection of unit jobs on two uniformly related machines. *J. Scheduling*, 17(1):87–93, 2014.

10 Naveen Garg and Amit Kumar. Better algorithms for minimizing average flow-time on related machines. In *ICALP*, volume 4051, pages 181–190. 2006.

11 Naveen Garg and Amit Kumar. Minimizing average flow-time : Upper and lower bounds. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 603–613, 2007.

12 Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 531–540, 2014.

13 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

14 Hans Kellerer, Thomas Tautenhahn, and Gerhard J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM J. Comput.*, 28(4):1155–1166, 1999.

15 Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *Journal of Computer and Systems Sciences*, 73(6):875–891, 2007.

16 Giorgio Lucarelli, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling in a resource augmentation model based on duality. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 63:1–63:17, 2016.

17 Giorgio Lucarelli, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online min-sum flow scheduling with rejections. In *In 13th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2017), 2017*, 2017.