2018 IEEE 59th Annual Symposium on Foundations of Computer Science

Improved Online Algorithm for Weighted Flow Time

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

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

***Abstract*—We discuss one of the most fundamental scheduling problem of processing jobs on a single machine to minimize the weighted flow time (weighted response time). Our main result is a $O(\log P)$-competitive algorithm, where P is the maximum-to-minimum processing time ratio, improving upon the $O(\log^2 P)$-competitive algorithm of Chekuri, Khanna and Zhu (STOC 2001). We also design a $O(\log D)$-competitive algorithm, where D is the maximum-to-minimum density ratio of jobs. Finally, we show how to combine these results with the result of Bansal and Dhamdhere (SODA 2003) to achieve a $O(\log(\min(P, D, W)))$-competitive algorithm (where W is the maximum-to-minimum weight ratio), without knowing P, D, W in advance. As shown by Bansal and Chan (SODA 2009), no constant-competitive algorithm is achievable for this problem.**

I. Introduction

We discuss the fundamental problem of online scheduling of jobs on a single machine. In this problem, a machine receives jobs over time, and the online algorithm decides which job to process at any point in time, allowing preemption as needed. Each of the jobs has a processing time (or volume) and a weight. We consider the weighted flow time cost function for this problem, in which the algorithm has to minimize the weighted sum over the jobs of the time between their arrival and their completion.

Define P as the maximal ratio between the processing times of any two jobs, and W as the maximal ratio between the weights of any two jobs. The first non-trivial algorithm for this problem was shown by Chekuri et al. [11], and is $O(\log^2 P)$-competitive. Bansal and Dhamdhere [6] showed a $O(\log W)$-competitive algorithm for this problem. A lower bound on competitiveness of $\Omega(\min(\sqrt{\log W / \log\log W}, \sqrt{\log\log P / \log\log\log P}))$ was then shown by Bansal and Chan [5].

An additional parameter of interest is D, which is the maximal ratio between the densities of any two jobs. This parameter has not been explored in any previous work, though the algorithm in [11] can be modified quite easily to yield a $O(\log^2 D)$-competitive algorithm.

This paper is partially supported by the ISF grant no. 1506/16 and by the ICRC Blavatnik Fund. A full version of this paper appears in https://arxiv.org/abs/1712.10273.

Our Results

Our main results are as follows:

- $O(\log P)$-competitive algorithm for weighted flow time on a single machine, improving upon the previous result of $O(\log^2 P)$ in [11].
- $O(\log D)$-competitive algorithm. This algorithm is different from the $O(\log P)$-competitive algorithm.
- $O(\log(\min(P, D, W)))$-competitive combined algorithm, without knowing P, D, W in advance. This builds on the previous two algorithms and the algorithm of [6].

Related Work

As mentioned above, for minimizing weighted flow time on a single machine, Chekuri et al. [11] presented a $O(\log^2 P)$-competitive algorithm. The algorithm arranges all jobs in a table by their weight and density. The algorithm only processes jobs the weight of which is larger than the sum of the weights in the rectangle of lower density and lower weight jobs. In [6], a $O(\log W)$-competitive algorithm is presented for this problem. This algorithm is based on the division of jobs into bins according to the logarithmic class of their weight, and then balancing the weights of those bins. A lower bound on algorithm competitiveness of $\Omega(\min(\sqrt{\log W / \log\log W}, \sqrt{\log\log P / \log\log\log P}))$ was shown in [5]. Typically, one assumes that P and W are bounded and that the number of jobs may be arbitrarily large (unbounded). In the case that the number of jobs is small, it is shown in [6] how to use the $O(\log W)$ competitive algorithm to construct a $O(\log P + \log n)$-competitive algorithm, with n being the number of jobs released. Note that the competitive ratio of this algorithm is unbounded as a function of P. For the offline problem, Chekuri and Khanna [10] showed a quasi-polynomial time approximation scheme. Bansal [4] later extended this result to a constant number of machines.

For a single machine in the unweighted case, a classic result from [16] states that the shortest remaining processing time algorithm is optimal. For the case of

2575-8454/18/$31.00 ©2018 IEEE
DOI 10.1109/FOCS.2018.00048

IEEE computer society

m identical machines, and a set of n jobs, Leonardi and Raz [15] showed that SRPT is $O(\log(\min(P, \frac{n}{m})))$-competitive, and that this is optimal. Awerbuch et al. [3] presented a $O(\log(\min(P, n)))$-competitive algorithm in which jobs do not migrate machines. An algorithm with immediate dispatching (and no migration) with similar guarantees was presented in [2]. In contrast to those positive results, for the case of weighted flow time on multiple machines, Chekuri et al. [11] showed a $\Omega(\min(\sqrt{P}, \sqrt{W}, (\frac{n}{m})^{\frac{1}{4}}))$ lower bound on competitiveness for $m > 1$ machines, making the problem intractable.

Competitive algorithms also exist for unweighted flow time on related machines [12], [13]. For the case of unweighted flow time for machines with restricted assignment, no online algorithm with a bounded competitive ratio exists [14].

In the resource augmentation problem, the single machine weighted flow time problem becomes significally easier. In [7], a $(1 + \frac{1}{\epsilon})$-competitive algorithm is given for a $(1+\epsilon)$-speed model. An algorithm with a similar guarantee was given in [6] for the non-clairvoyant setting. A competitive algorithm is also known for weighted flow time in unrelated machines [8]. Additional work has been done on weighted flow time for unrelated machines with resource augmentation in the offline model (see e.g. [9], [1]).

Our Technique

Processing-time ratio algorithm. Our $O(\log P)$ algorithm rounds the weights of incoming jobs to a power of 2, and assigns them into power-of-2 bins according to their processing time. That is, bin i contains jobs whose initial processing time is in the range $[2^i, 2^{i+1})$. Note that jobs never move between bins.

To describe the algorithm, we utilize a visualization of the algorithms state, as shown in figure 1. We view each job as a rectangular container, such that the height of the rectangle is the weight of the job, and its area is 2^{i+1}. The volume of the job can be viewed as liquid within that container. When a job is being processed, the amount of liquid is reduced from the top. A horizontal dotted line runs through the middle of the container. This line helps to observe the volume covered by a bar, a concept used in the algorithm's analysis.

Inside each bin, jobs of higher weight have a higher priority for processing. For jobs of the same weight, the jobs are ordered according to their density, such that lower density jobs get priority. This priority is illustrated by the rectangular containers of the jobs being stacked on top of one another, such that a higher job in the stack has higher priority.

At any point in time, the algorithm chooses a bin from which to process the uppermost job. At each point in time, each bin is assigned a score, such that the bin with the highest score is processed. In [6], the score assigned to each bin is the total weight of the jobs in that bin. In our algorithm, the score is more complex. All jobs except the top job add their weight to the total score of the bin. The top job adds to the score of the bin either its complete weight, if it has high remaining processing time, or half of it, if it has low remaining processing time. While this modification seems odd, it is crucial for the proof. This can lead to some interesting behavior: for example, a job can be preempted during processing because its processing time has decreased below the threshold, lowering the score of its bin. This preemption is not due to any external event; no job has been released to trigger it.

The outline of our proof is inspired by [6]. In our proof, we use the concept of volume covered by a bar. We place a horizontal bar at some height x, as shown in figure 1. If the bar lies over the top of a job, we say that it covers the entire volume of the job. If the bar lies between the half of the job (the dotted lines in figure 1) and the top of the job, it only covers the volume underneath the half of the job. Otherwise, the bar covers none of the job's volume.

The proof consists of showing that at any time, a bar at a height that is constant times the remaining weight of the optimum, every job in the algorithm covers the entire volume in the algorithm. We then show that such a bar can be raised by a multiplicative factor of 2 to yield a second bar, which covers all weight in all bins (and therefore, its height is larger than the total weight in that bin). This gives us the $O(\log P)$ competitiveness.

Density ratio algorithm. A different algorithm, though similar in the method of its proof, is used for $O(\log D)$ competitiveness. We assign the jobs into power-of-2 bins according to density. In this algorithm, the weights of the algorithm are not rounded to a power of 2. Instead, the densities of the jobs are rounded to a power of 2 (this is also through changing the weights). This gives us that bin i contains jobs whose initial density is 2^i. In this algorithm, as in the previous algorithm, jobs never move between bins.

A visualization of a possible state of the algorithm at a time t is shown in figure 4. We again have a rectangular container for each job, the height of which is the weight of the job. In this case, the container arrives full–its width is 2^i, and thus its area is exactly the volume of the job upon its arrival. As in the $O(\log P)$ algorithm, when a job is processed its container depletes from the top. Inside each bin, one

job takes priority over another if its weight is of a higher power of 2 (higher weight class). For jobs of the same weight class, a partially processed job takes priority.

In this algorithm, each bin has a single dummy job of each weight class. To calculate the score of a bin, we observe its top (non-dummy) job. The jobs below that top job–dummy or not–add their weight to the score. The top job, however, adds to the score only a fraction of its weight, which is the fraction of its volume that has not yet been processed. Thus, the score of a bin depends continuously on the processing state of the top job, as opposed to the two discrete options in the $O(\log P)$ algorithm.

This score again stems from a definition for the volume covered by a bar. In this case, the definition is continuous–the volume covered by the bar is exactly the volume that appears under the bar in a visualization such as figure 4.

Combined algorithm. Finally, we use the fact that the $O(\log W)$ algorithm of [6] and the $O(\log P)$ and $O(\log D)$ algorithms have some common properties, and show how to combine them into a $O(\log(\min(P, D, W)))$ competitive algorithm without knowledge of P, D, W in advance. This relies on bins of three types coexisting, and being assigned jobs in a manner that prefers bins that have already been in use. We note that this method works by using the specific common properties of the different algorithms, and cannot be applied to general algorithms for these three parameters.

II. Preliminaries

A. The Model

We are given a single machine, and jobs arrive over time. The machine can choose which job to process at any given time, with the option to preempt a previous job if necessary.

An instance in the model is a set of jobs $\mathcal{J}$, so that every job $J \in \mathcal{J}$ of index $I(J) \in \mathbb{N}$ has the following attributes:

- A processing time $p(J) > 0$ (also called the volume of J).
- A weight $w(J) > 0$, which represents the significance of the job.
- A time of release $r(J) \geq 0$. The job J cannot be processed prior to this time.

Let the density of a job J be $d(J) = \frac{p(J)}{w(J)}$. We consider the online clairvoyant model, in which an algorithm is not aware of the existence of a job J of index $I(J)$ at time $t < r(J)$, but knows $p(J), w(J)$ once $t \geq r(J)$.

For a given algorithm and an instance $\mathcal{J}$, denote by $c(J)$ the completion time of a job $J \in \mathcal{J}$ in the algorithm. The goal of the algorithm is to minimize weighted flow time, defined as:

$$\mathscr{C}(\mathcal{J}) = \sum_{J \in \mathcal{J}} w(J) \cdot (c(J) - r(J))$$

Note that when all $w(J) = 1$ this reduces to the flow time of the algorithm, which is the term $\sum_{J \in \mathcal{J}} (c(J) - r(J))$.

For a specific instance $\mathcal{J}$, we define the parameters $P = \max_{J_1, J_2 \in \mathcal{J}} \frac{p(J_1)}{p(J_2)}$, $W = \max_{J_1, J_2 \in \mathcal{J}} \frac{w(J_1)}{w(J_2)}$ and $D = \max_{J_1, J_2 \in \mathcal{J}} \frac{d(J_1)}{d(J_2)}$. Our algorithms do not require knowledge of these parameters in advance.

B. Common Definitions

All algorithms we propose consist of two parts. The first part assigns every job to a bin upon release. The second part decides which job to process at any given time, based on the contents of those bins.

Denoting by $\mathcal{A}$ the set of bins used by our algorithm, we use the following definitions.

Definition II.1. Define the following:

- For every bin $A \in \mathcal{A}$ and a point in time t, define $A(t)$ as the set of jobs alive in the algorithm in A at t (i.e. jobs that have arrived but have not yet been completed).
- Denote by $Q(t) = \bigcup_{A \in \mathcal{A}} A(t)$ the set of all jobs alive at time t in the algorithm.
- For a set of jobs S, define $w(S) = \sum_{J \in S} w(J)$. Define $W(t) = w(Q(t))$
- Define $p_t(J)$ to be the remaining processing time of J (also called the remaining volume of J) at time t.
- For a set of jobs S alive at time t, define $p_t(S) = \sum_{J \in S} p_t(J)$. Define $V(t) = p_t(Q(t))$.

When considering an optimal algorithm, we refer to its attributes by adding an asterisk to the aforementioned notation. For example, $W^*(t)$ is the living weight in the optimal algorithm at time t, and $p_t^*(J)$ is the remaining processing time of job J in the optimal algorithm at time t.

Our proofs use the following observation to show competitiveness.

Observation II.2. *If $W(t) \leq c \cdot W^*(t)$ for all t then the algorithm is c-competitive. This is due to the fact that the weighted flow time of the algorithm can be expressed as $\int_0^\infty W(t)dt$.*

III. The $O(\log P)$-Competitive Algorithm

In this section, we present a $O(\log P)$-competitive algorithm for weighted flow time. The algorithm consists

of two parts. The first part, assigns incoming jobs immediately to a bin from the set $\mathcal{A} = \{A_i \mid i \in \mathbb{Z}\}$. The second part processes jobs from the bins.

The algorithm **assumes wlog** that every job J arrives with a weight that is an integral power of 2. The algorithm can enforce this by rounding the weight of every job J to $2^{\lg(w(J))+1}$, which adds a factor of 2 to its competitive ratio.

Within each bin, we have an ordering between jobs that determines their priority.

Definition III.1. At a time t and bin A:

- For J_1, J_2 in $A(t)$, we write $J_1 \prec_t J_2$ when $w(J_2) > w(J_1)$ or when $w(J_2) = w(J_1)$ and $p_t(J_1) > p_t(J_2)$. If $w(J_2) = w(J_1)$ and $p_t(J_1) = p_t(J_2)$, we break ties according to the indices of the jobs, $I(J_1)$ and $I(J_2)$ (i.e. arbitrarily).
- We denote by $\text{top}_A(t)$ the maximal job with respect to $\prec_t$.

Definition III.2. For a bin $A_i \in \mathcal{A}$ and a time t:

- For a job J assigned to A_i, we say that J is *well-processed* if $p_t(J) \leq 2^i$. Denote by $\delta(J)$ the indicator variable for being well-processed.
- Define the *score* of bin A_i to be $\mathscr{W}_{A_i}(t) = w(A_i(t)) - \delta(\text{top}_{A_i}(t)) \cdot \frac{w(\text{top}_{A_i}(t))}{2}$.

Algorithm 1 as described below is $O(\log P)$ competitive.

```
1 Whenever a new job J arrives:
2     assign J to A_i such that
        2^i < p(J) ≤ 2^{i+1}.
3 At any time t:
4     For A = arg max_A(W_A(t)), process
        top_A(t).
```

Algorithm 1: $O(\log P)$ Competitive

IV. Analysis of $O(\log P)$-Competitive Algorithm

Consider a visualization such as figure 1, of the state of the algorithm at a time t. The *base* of a job $J \in A_i(t)$ (denoted $\beta(J,t)$) is the height of its bottom in the visualization. Formally, $\beta(J,t) = w(\{J' \in A_i(t) \mid J' \prec_t J\})$.

Consider a horizontal bar at height x. We now define *the volume covered by* x, or *the volume under* x. This is:

- All the volume of jobs completely under x.
- None of the volume of jobs completely over x.
- Some of the volume of jobs that intersect x.

Formally:

Definition IV.1. At any time t, and a bin $A_i \in \mathcal{A}$:

- For any job $J \in A_i(t)$ and a non-negative x, the *volume of J under x* is:

$$\gamma_J(x,t) = \begin{cases} p_t(J) & x \geq \beta(J,t) + w(J) \\ 0 & x < \beta(J,t) + \frac{w(J)}{2} \\ \min(p_t(J), 2^i) & \beta(J,t) + \frac{w(J)}{2} \leq x \\ & < \beta(J,t) + w(J) \end{cases}$$

- The *volume of bin A_i under x* is:

$$B_{A_i}(x,t) = \sum_{J \in A_i(t)} \gamma_J(x,t)$$

- The *total volume under x* is:

$$B(x,t) = \sum_{A_i \in \mathcal{A}} B_{A_i}(x,t)$$

The previous definition yields the following observation.

Observation IV.2. *$\mathscr{W}_{A_i}(t)$ as defined before is the minimal bar that covers the entire volume of a bin A_i.*

In this section, we prove the following theorem.

Theorem IV.3. *The algorithm described in Section III is $O(\log P)$-competitive.*

To prove Theorem IV.3, we first prove Lemmas IV.5 and IV.8. We then show that those lemmas, together with Proposition IV.9, imply the competitiveness of the algorithm.

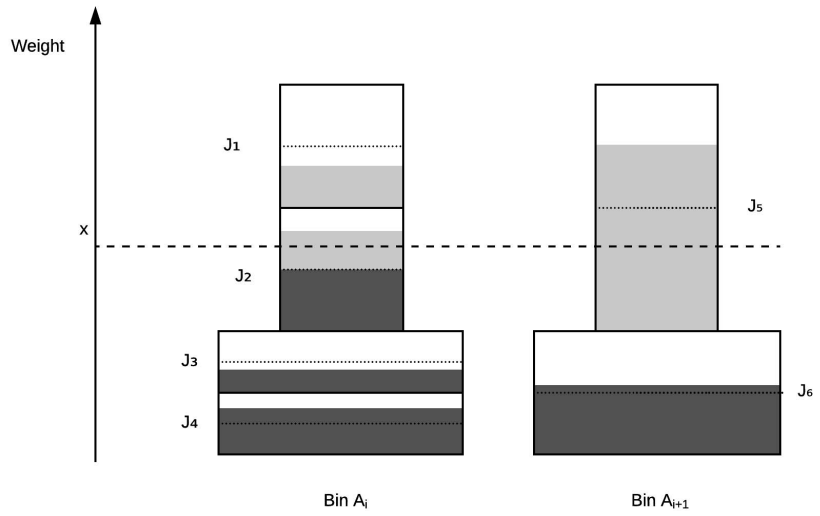
Lemma IV.5 states that the volume of a bin that is covered by a bar does not decrease upon the arrival of a new job at that bin. Lemma IV.8 states that when a job arrives at a bin, raising the bar by three times the weight of that new job is enough in order to gain its volume. A visualization of lemma IV.5 is given in figure 2.

Observation IV.4. *If at some point in time t we have that $J_1 \prec_t J_2$, then $J_1 \prec_{t'} J_2$ at any time t' in which both J_1 and J_2 are alive. This is since processing a job can only increase its priority, and the algorithm always processes a job of maximum priority. We can therefore write $J_1 \prec J_2$.*

Lemma IV.5. *If a job J_0 is released at time t and assigned to A_i, and defining t^- to be the time t prior to the event of J_0's release, then for every non-negative x we have that*

$$B_{A_i}(x,t) \geq B_{A_i}(x,t^-)$$

The following image is a possible state of two bins in the algorithm at some time t.



In this figure, each job is described by a rectangular container. The height of the container is the weight of the job, and its area is 2^{j+1} for bin A_j. The area of the gray rectangles inside each container represent the volume of the job. The container is separated into two halves by a dotted line.
The jobs are arranged in the bins according to $\prec$, with $J_4 \prec J_3 \prec J_2 \prec J_1$ and $J_6 \prec J_5$. $\beta(J,t)$ is the height of J in this figure - for example, $\beta(J_2,t) = w(J_3) + w(J_4)$.
The volume under x is illustrated by a darker gray. For example:

- For J_3 we have $\beta(J_3,t) + w(J_3) = w(J_4) + w(J_3) \leq x$ and therefore $\gamma_{J_3}(x,t) = p_t(J_3)$.
- For J_2 we have that $\beta(J_2,t) + \frac{w(J_2)}{2} \leq x < \beta(J_2,t) + w(J_2)$ giving that $\gamma_{J_2}(x,t)$ is the area of J_2 under the dotted line. In other words, it is $\min(2^i, p_t(J_2)) = 2^i$.
- For J_5 we have that $\beta(J_5,t) + \frac{w(J_5)}{2} > x$, which yields $\gamma_{J_5}(x,t) = 0$.

As for processing, note that the algorithm will process bin A_{i+1} and not A_i, though their weights are equal. This is since J_1 adds only $\frac{w(J_1)}{2}$ to $\mathscr{W}_{A_i}(t)$, because J_1's volume is under its dotted line. Since the volume of J_5 is above J_5's dotted line, it adds its full weight, $w(J_5)$, to $\mathscr{W}_{A_{i+1}}(t)$.

Figure 1: Processing Time Bins

Proof: Denote by $L \subseteq A_i(t)$ the set of jobs J' such that $\gamma_{J'}(x,t^-) > \gamma_{J'}(x,t)$. Note that for every $J' \in L$ we have $J_0 \prec J'$, and thus $w(J') \geq w(J_0)$.

For every $J' \in L$, we define $\Delta_{J'} = \gamma_{J'}(x,t) - \gamma_{J'}(x,t^-)$ (note that from the definition of L, we have that $\Delta_{J'} < 0$). It is enough to show

$$\gamma_{J_0}(x,t) + \sum_{J' \in L} \Delta_{J'} \geq 0 \tag{IV.1}$$

Define $J_{\max}$ the maximal job in L at time t with respect to $\prec_t$. We seperate into two cases.

Case 1: If $w(J_{\max}) > w(J_0)$, then due to the weights being powers of 2 we must have $w(J_{\max}) \geq 2w(J_0)$. In this case, the arrival of J_0 either:

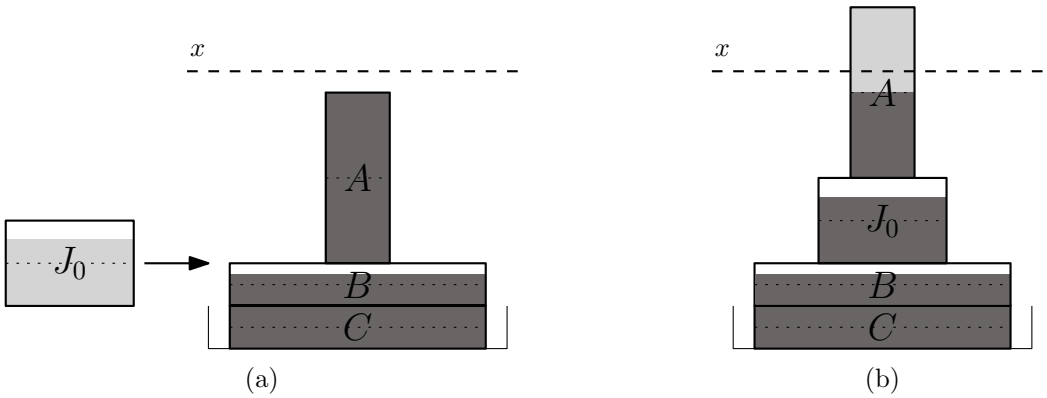
- Changed $J_{\max}$ from being completely covered by x at time t, to having x at least half the height of $J_{\max}$, but below the top of $J_{\max}$.
- Changed $J_{\max}$ from having x at least half the height of $J_{\max}$, to being below half the height of $J_{\max}$ but at least $J_{\max}$'s base.

In either case, $J_{\max}$ has at most 2^i less volume covered by x at time t than at t^-. In addition, since x intersects $J_{\max}$ at time t, all volume of jobs below $J_{\max}$ is covered by $J_{\max}$. This yields $L = \{J_{\max}\}$, as well as $\gamma_{J_0}(x,t) = p(J_0)$. This completes case 1.

Case 2:

Assume that $w(J_{max}) = w(J_0)$. Since $J_0 \prec J_{\max}$, we have that $p_t(J_{\max}) \leq p_t(J_0) = p(J_0)$. We separate into the following subcases:

- Suppose x is at least $J_{\max}$'s top at time t^-. It must



The two images above, image (a) and image (b), show a bin before and after the arrival of J_0. In both (a) and (b) we have a bar at height x, with the volume covered by the bar colored in dark gray. Lemma IV.5 states that there is at least as much dark gray volume in image (b) as in image (a).

Figure 2: Visualization of Lemma IV.5

be at least $J_{\max}$'s bottom, and below $J_{\max}$'s top, at time t. Since x intersects $J_{\max}$ at time t, we have $L = \{J_{\max}\}$ and $\gamma_{J_0}(x,t) = p(J)$. This gives us that $\Delta_J \geq -p_t(J_{\max}) \geq p(J)$, completing this subcase.

- Suppose x is at least $\beta(J_{\max}, t^-) + \frac{w(J_{\max})}{2}$ and below $J_{\max}$'s top at time t^-. Then at time t, $\beta(J_{\max}, t) - \frac{w(J_0)}{2} \leq x < \beta(J_{\max}, t)$. Denoting by $J' \in A_i(t)$ the job directly below $J_{\max}$ at time t, we have that x intersects J' at time t.
  - If $J' = J_0$, then $L = \{J_{\max}\}$. $J_{\max}$ lost its lower half, which is at most 2^i volume. However, J_0's lower half is covered, and $p(J_0) \geq 2^i$. Therefore, $\gamma_{J_0}(x,t) = 2^i$ as required.
  - If $J' \neq J_0$, then $L \subseteq \{J_{\max}, J'\}$. x lost $J_{\max}$'s lower half, at most 2^i volume. Since $p(J_0) \geq 2^i$, this is at most the volume of J_0's lower half. It also lost J''s upper half, which has less volume than J_0's upper half (recall that $p(J_0) \geq p_t(J')$). However, since x intersects J' at time t, $\gamma_{J_0}(x,t) = p(J_0)$, as required.

This completes case 2, and the lemma. ■

Proposition IV.6. *At every time t and bin A_i, there exists at most one well-processed job $J \in A_i(t)$ of each weight.*

Proof: Since no job arrives well-processed, every job must become well-processed through processing. For any possible weight 2^j, once a job J of that weight becomes well-processed the algorithm does not process any non-well-processed job of the same weight in A_i until J is complete. This implies the lemma. ■

Corollary IV.7. *At every time t, bin A_i and $j \in \mathbb{Z}$, let $S \subseteq A_i(t)$ be the subset of well-processed jobs of weight at most 2^j. Then $w(S) < 2^{j+1}$.*

Proof: Let $j_{\min}$ the minimal index such that S has a job of weight $2^{j_{\min}}$. Using Proposition IV.6:

$$w(S) \leq \sum_{k=j_{\min}}^{j} 2^k < 2^{j+1}$$

■

Lemma IV.8. *If a job J_0 is released at time t and assigned to A_i, and defining t^- to be the time t prior to the event of J_0's release, then for every non-negative x we have that*

$$B_{A_i}(x + 3 \cdot w(J_0), t) \geq B_{A_i}(x, t^-) + p(J_0)$$

Proof: Note that for every job $J' \in A_i(t^-)$, the base of J' can increase by at most $w(J_0)$ upon the arrival of J_0. Thus, since the bar x is also raised by $3 \cdot w(J_0)$ (more than $w(J_0)$), we have $\gamma_{J'}(x + 3 \cdot w(J_0), t) \geq \gamma_{J'}(x, t^-)$. Therefore, it is enough to find a subset $S \subseteq A_i(t)$ such that the volume of S covered by $x + 3 \cdot w(J_0)$ at t is at least $p(J_0)$ more than the volume of S covered by x at t^-.

We observe the position of J_0 at time t. If J_0 is covered by $x + 3w(J_0)$, that is $\beta(J_0, t) \leq x + 2w(J_0)$, then $\gamma_{J_0}(x + 3w(J_0), t) = p(J_0)$. Noting that J_0 did not exist at t^-, we choose $S = \{J_0\}$ and the proof is complete.

Otherwise, assume $\beta(J_0, t) > x + 2w(J_0)$. We consider the set of jobs that begin and end in the interval $[x, x + 3w(J_0)]$ – that is, jobs J' such that $\beta(J', t) \geq x$ and $\beta(J', t) + w(J') \leq x + 3w(J_0)$. Each job $J' \in S$ has the

property that $\gamma_{J'}(x, t^-) = 0$ and $\gamma_{J'}(x + 3w(J_0), t) = p_t(J')$. To complete the proof, it only remains to be seen that $p_t(S) \geq p(J_0)$.

Observe that the interval $[x, x + 3w(J_0)]$ can only contain jobs from the set S, and possibly some part of the job immediately below S (denoted $J_\perp$) and the job immediately above S (denoted $J_\top$). Therefore, we must have that $w(S) \geq 3w(J_0) - w(J_\perp) - w(J_\top)$.

Since S lies wholly below J_0, we must have that $J_\perp, J_\top$ have weight at most $w(J_0)$ – thus, S is non-empty. We observe two cases, based on the maximal weight of a job in S.

Case 1: $\max_{J \in S} w(J) = w(J_0)$

S contains a job J' of weight $w(J_0)$, but J' is below J_0, yielding $p_t(J') \geq p_t(J_0) = p(J_0)$ and completing the case.

Case 2: $\max_{J \in S} w(J) \leq \frac{w(J_0)}{2}$

In this case, we show that S contains two jobs that are not well-processed, yielding $p_t(S) \geq 2^{i+1} \geq p(J_0)$ and completing the proof. $J_\perp$ is below S, and thus $w(J_\perp) \leq \frac{w(J_0)}{2}$, yielding $w(S) \geq \frac{3w(J_0)}{2}$. However, due to Lemma IV.7, the total weight of well-processed jobs of weight at most $\frac{w(J_0)}{2}$ is less than $\frac{w(J_0)}{2}$. Thus, the total weight of the jobs of S that are not well-processed is more than $\frac{w(J_0)}{2}$, which means at least two jobs.

■

We now show that Lemmas IV.5 and IV.8 imply the competitiveness of the algorithm. The outline of the proof is shown in figure 3, which shows the state of the algorithm (on the left) and the optimum (on the right) at any time t. First, Lemma IV.11 shows that the bar $3 \cdot W^*(t)$ covers the entire volume in the algorithm. Lemma IV.12 then uses that fact to show that the bar $6 \cdot W^*(t)$ covers the entire *weight* in the algorithm (in the figure, it covers the top of each bin). The fact that there are $O(\log P)$ bins then implies competitiveness.

The following proposition makes use of the choice of bin made by the algorithm.

Proposition IV.9. *At a time t and a non-negative x, if $B(x, t)$ decreases as a result of the algorithm's processing, then $B(x, t) = V(t)$.*

Proof: Note that $\mathscr{W}_{A_i}(t)$ as defined before is exactly the minimal bar that covers all volume in A_i (or equivalently, all volume of the top job in A_i). Also note that a bar can only lose volume from the processing of a job if it covers the entire volume of that job.

Let x be such that $B(x, t)$ decreases through processing. Since the job being processed is the top job of some bin A_i, this implies $x \geq \mathscr{W}_{A_i}(t)$. From the algorithm's definition, $\mathscr{W}_{A_i}(t) \geq \mathscr{W}_{A_{i'}}(t)$ for all $A_{i'} \in \mathcal{A}$, yielding that x covers all volume in $A_{i'}$. Summing, we have $B(x, t) = V(t)$. ■

Definition IV.10. For any two points in time t, t' such that $t' \in [0, t]$, let $Q_t(t')$ be the subset of $Q(t)$ that was alive at time t'. Formally, $Q_t(t') = Q(t) \cap Q(t')$.

Lemma IV.11. *At every time t, a bar at three times the weight of the optimum covers at least the volume of the optimum. Formally:*

$$V^*(t) \leq B(3 \cdot w(Q^*(t)), t)$$

Proof: To show the lemma, we show that for every $t' \in [0, t]$ we have:

$$p^*_{t'}(Q^*_t(t')) \leq B(3 \cdot w(Q^*_t(t')), t') \qquad (*)$$

When $t' = t$, Inequality (*) yields the lemma. We show (*) by induction, as t' advances from 0 to t.

We can see that for $t' = 0$, before the release of any jobs, (*) is trivially true.

Now consider all k jobs that arrive in the interval $[0, t]$, and denote their release times by $t_1, ..., t_k$, so that in time t_i the i'th job has already been released. For a job i, let t_i^- be the time t_i before the event of the release of the i'th job. For convenience, denote $t_0 = 0$ and $t^-_{k+1} = t$. We show the following claims:

Claim 1: If (*) holds for $t' = t_i$ such that $0 \leq i \leq k$, then (*) holds for every $t' \in [t_i, t^-_{i+1}]$.

Since no job arrives in that interval, we have that $p^*_{t'}(Q^*_t(t'))$ is non-increasing as t' increases. It remains to consider all cases in which $B(3 \cdot w(Q^*_t(t')), t')$ decreases as a result of the algorithm's processing. Let t' be such that $B(3 \cdot w(Q^*_t(t')), t')$ decreases as a result of execution. Then using Proposition IV.9, we have that $B(3 \cdot w(Q^*_t(t')), t') = V(t') = V^*(t') \geq p^*_{t'}(Q^*_t(t'))$ as required.

Claim 2: If (*) holds for $t' = t_i^-$ such that $1 \leq i \leq k$, then (*) holds for $t' = t_i$.

We'll observe the following cases.

Case 1: The i'th job, denoted J, is such that $J \notin Q^*_t(t_i)$. we have that $Q^*_t(t_i) = Q^*_t(t_i^-)$, and thus

$$\begin{aligned} B(3 \cdot w(Q^*_t(t_i)), t_i^-) &= B(3 \cdot w(Q^*_t(t_i^-)), t_i^-) \\ &\geq p^*_{t_i^-}(Q^*_t(t_i^-)) = p^*_{t_i}(Q^*_t(t_i)) \end{aligned}$$

Denoting by $A \in \mathcal{A}$ the bin to which J has been assigned, and using Lemma IV.5, we have that $B_A(3 \cdot w(Q^*_t(t_i)), t_i) \geq B_A(3 \cdot w(Q^*_t(t_i)), t_i^-)$. As for any other bin $A' \in \mathcal{A}$, since A' has not changed upon the release of J we must have that $B_{A'}(3 \cdot w(Q^*_t(t_i)), t_i^-) = B_{A'}(3 \cdot w(Q^*_t(t_i)), t_i)$. This gives us

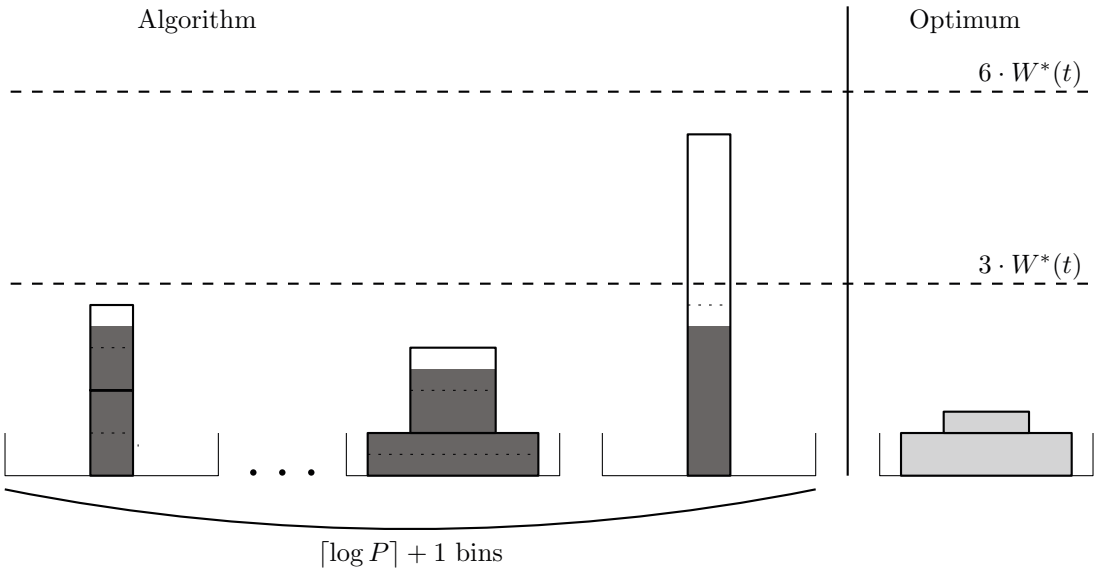


Figure 3: Visualization of Competitiveness Argument

that $B(3 \cdot w(Q_t^*(t_i)), t_i) \geq B(3 \cdot w(Q_t^*(t_i)), t_i^-)$, and thus the claim for this case.

Case 2: The i'th job, denoted J, is such that $J \in Q_t^*(t_i)$. we have that $Q_t^*(t_i) = Q_t^*(t_i^-) \cup \{J\}$, and thus

$$\begin{aligned} B(3 \cdot w(Q_t^*(t_i^-)), t_i^-) &\geq p^*_{t_i^-}(Q_t^*(t_i^-)) \\ &= p^*_{t_i}(Q_t^*(t_i)) - p(J) \end{aligned}$$

Denoting by $A \in \mathcal{A}$ the bin to which J has been assigned, using Lemma IV.8, we have that $B_A(3 \cdot w(Q_t^*(t_i)), t_i) = B_A(3 \cdot w(Q_t^*(t_i^-)) + 3 \cdot w(J), t_i) \geq B_A(3 \cdot w(Q_t(t_i^-)), t_i^-) + p(J)$. As for any other bin $A' \in \mathcal{A}$, as in the previous case we must have that $B_{A'}(3 \cdot w(Q_t^*(t_i^-)), t_i^-) = B_{A'}(3 \cdot w(Q_t^*(t_i^-)), t_i) \leq B_{A'}(3 \cdot w(Q_t^*(t_i)), t_i)$. Summing over the bins gives us the claim for this case.

Claims 1 and 2 show (*) for every $t' \in [0, t]$, completing the proof. ∎

Lemma IV.12. *Denoting by $\mathcal{A}' \subset \mathcal{A}$ the set of non-empty bins at a time t, we have:*

$$W(t) \leq 6 \cdot |\mathcal{A}'| \cdot W^*(t)$$

Proof: For every time t, using Lemma IV.11, we have that $B(3 \cdot w(Q^*(t)), t) \geq V^*(t)$

Now, we can assume without loss of generality that that the optimal algorithm is never idle when a job remains uncompleted. This implies $V(t) = V^*(t)$, and thus $B(3 \cdot w(Q^*(t)), t) = V(t)$.

Denoting $x = 3 \cdot w(Q^*(t))$, for every bin $A \in \mathcal{A}$, observe $J = \text{top}_A(t)$. Since x covers all of J's volume, and thus $x \geq \beta(J, t) + \frac{w(J)}{2} = w(A(t) \backslash \{J\}) + \frac{w(J)}{2}$. Thus, we have that $2x \geq 2w(A(t) \backslash \{J\}) + w(J) \geq w(A(t))$. Therefore, $w(A(t)) \leq 6 \cdot w(Q^*(t))$, and summing over all $\mathcal{A}'$, $W(t) \leq 6 \cdot |\mathcal{A}'| \cdot W^*(t)$. ∎

We can now prove the main theorem.

Proof: (of Theorem IV.3) The algorithm assigns jobs to at most $\lceil \log P \rceil + 1$ bins. Using Lemma IV.12, we therefore have that:

$$W(t) \leq 6(\lceil \log P \rceil + 1) \cdot W^*(t)$$

This gives us that the algorithm is $O(\log P)$-competitive. ∎

V. The $O(\log D)$-Competitive Algorithm

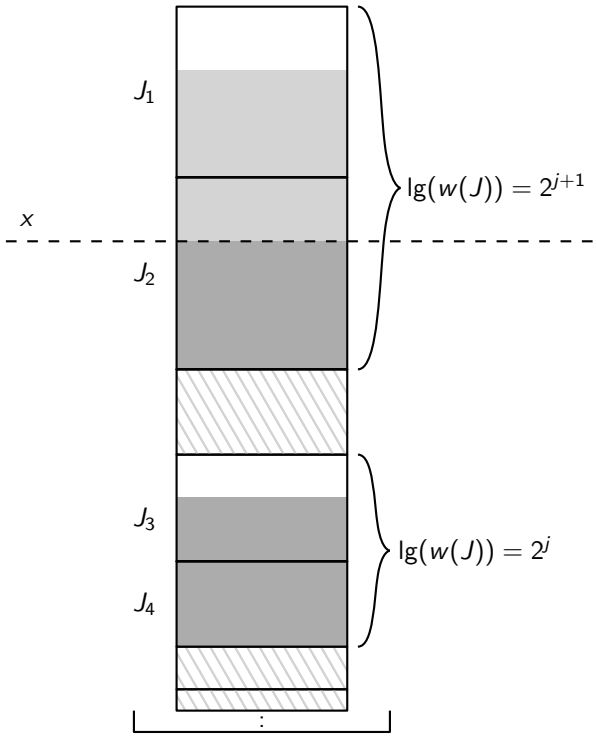
We now describe an $O(\log D)$-competitive algorithm. As in Section III, this algorithm has an assignment part and a processing part. The bins themselves are the set $\mathcal{A} = \{A_i \mid i \in \mathbb{Z}\}$.

This algorithm makes the assumption that every job J arrives so that for some integer i we have $d(J) = 2^i$. This assumption can be enforced by rounding the

```
1 Whenever a new job J arrives:
2     assign J to A_{log_2(d(J))}
3 At any time t:
4     For A = arg max_A(W'_A(t)), process
        top_A(t).
```

Algorithm 2: $O(\log D)$ Competitive

The following image is a possible state of a bin in the algorithm at some time t.



In this figure, the dummy jobs are dashed with gray lines. In the base of the bin is the infinite base of dummy jobs, the height of which sums to 2^j. The 4 real jobs, J_1 through J_4, are stacked in the bin according to $\prec$, such that $J_4 \prec J_3 \prec J_2 \prec J_1$. Each real job is again represented as a container, in which the gray area is the remaining processing time of the job. Its height is the weight of the job, and its width is 2^i for the bin A_i. Since in A_i all jobs have density 2^i, the area of the container is exactly the initial volume of the job - that is, all containers arrive full.
A bar x is shown in the figure. The dark gray area in the figure is the volume covered by the bar x, which changes continuously with x (compare this with covered volume in Section IV, which has discrete "thresholds").

Figure 4: Density Bins

weight of each incoming job up by a factor of at most 2 to give the desired density. This only adds a factor of 2 to the competitive ratio of the algorithm, similarly to Section III.

Definition V.1. For some positive x, define $\lg x = \lfloor \log_2 x \rfloor$.

We now redefine the ordering $\prec_t$ within a bin.

Definition V.2. For every bin A at time t, and for distinct $J_1, J_2 \in A(t)$, we write $J_1 \prec_t J_2$ if $\lg(w(J_1)) < \lg(w(J_2))$, or if $\lg(w(J_1)) = \lg(w(J_2))$ and J_2 is partially processed. If $\lg(w(J_1)) = \lg(w(J_2))$ and both J_1, J_2 are not partially processed, we break ties according to the indices of the jobs, $I(J_1)$ and $I(J_2)$ (i.e. arbitrarily).

Note that the previous definition did not address the case of $\lg(w(J_1)) = \lg(w(J_2))$ and J_1, J_2 are *both* partially processed. This case never happens – once there is a single partially processed job J in a specific weight class, the algorithm will not process another job of that class from the same bin until J is complete. Thus, a bin cannot contain more than one partially processed job in any specific weight class.

We recall the definition $\text{top}_A(t)$ as the maximal job in bin A at time t with respect to $\prec_t$.

Definition V.3. For a bin $A = A_i \in \mathcal{A}$ and a time t, define

$$\mathscr{W}'_A(t) = w(A(t) \backslash \{\text{top}_A(t)\}) + 2^{\lg(w(\text{top}_A(t)))} + \frac{p_t(\text{top}_A(t))}{2^i}$$

When a new job J arrives:
  if A_i *such that* $2^i < p(J) \leq 2^{i+1}$ *is open* **then**
    assign J to A_i
    round the weight of J up to $2^{\lg(w(J))+1}$
  else if $A'_{\lg(d(J))}$ *is open* **then**
    assign J to $A'_{\lg(d(J))}$
    round the weight of J up to give J the new density $2^{\lg(d(J))}$
  else if $A''_{\lg(w(J))+1}$ *is open* **then**
    assign J to $A''_{\lg(w(J))+1}$
    round the weight of J up to $2^{\lg(w(J))+1}$
  else
    open A_i, $A'_{\lg(d(J))}$ and $A''_{\lg(w(J))+1}$
    assign J to $A''_{\lg(w(J))+1}$
    round the weight of J up to $2^{\lg(w(J))+1}$
At any time t:
  For $A = \arg\max_A(\tilde{\mathscr{W}}_A(t))$, process $\text{top}_A(t)$

Algorithm 3: $O(\log(\min(P, D, W)))$ Competitive

Algorithm 2 is $O(\log D)$ competitive, as discussed in the next section.

VI. Analysis of $O(\log D)$-Competitive Algorithm

The following theorem states the competitiveness of Algorithm 2.

Theorem VI.1. *Algorithm 2 is $O(\log D)$-competitive.*

The proof of Theorem VI.1 is given in the full version of the paper on arXiv. The outline of the proof is similar to that of Theorem IV.3, using a modified definition of covered volume, which we now describe.

For the purpose of viewing the volume covered by a bar, we add a dummy job of each weight class to each bin. That is, a dummy job of weight 2^i is added for any integer i. Each dummy job has higher priority than (i.e. lies above) any other job of its weight class. Note that for each real (non-dummy) job J, the total weight of dummy jobs below J sums to $2^{\lg(w(J))}$. The dummy jobs have no volume, are never processed and are only for the purpose of modifying the base of real jobs.

As in the processing time algorithm, we define the base of a job J at time t to be $\beta(J,t) = w(\{J' \in A_i(t) \mid J' \prec_t J\})$.

We now give the modified definition of the volume covered by a bar. Informally, a bar x covers exactly the volume that appears underneath x in the visualization. Formally:

Definition VI.2. The volume of job J under bar x is:

$$\gamma_J(x,t) = \begin{cases} p_t(J) & x \geq \beta(J,t) + \frac{p_t(J)}{2^i} \\ 2^i(x - \beta(J,t)) & \beta(J,t) \leq x \\ & < \beta(J,t) + \frac{p_t(J)}{2^i} \\ 0 & \text{otherwise} \end{cases}$$

A visualization of a density bin with dummy jobs, as well as the new definition of covered volume, can be found in Figure 4.

VII. The $O(\log(\min(W, P, D)))$-Competitive Algorithm

In this section we describe an algorithm which is $O(\min(\log(W, P, D)))$-competitive without knowing W, P, D in advance. As in Sections III and V, the algorithm is composed of a bin-assignment part and a bin-processing part.

The main idea in the algorithm is combining bins for processing time, bins for density and bins for weight. In this algorithm, all the bins start closed, and must be opened prior to being assigned any jobs by the algorithm. The algorithm only opens bins as triplets with a bin of each type; this property keeps the number of bins logarithmic in the minimal of W, P, D.

Define the disjoint sets of bins $\mathcal{A}_1 = \{A_i \mid i \in \mathbb{Z}\}$ (processing time bins), $\mathcal{A}_2 = \{A'_i \mid i \in \mathbb{Z}\}$ (density bins) and $\mathcal{A}_3 = \{A''_i \mid i \in \mathbb{Z}\}$ (weight bins). Our set of bins is $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3$.

Inside the processing-time bins and density bins, the jobs are ordered as in Sections III and V respectively. Inside weight bins, the jobs are ordered according to remaining processing time (lower processing time is higher).

Since the algorithm uses various ways of rounding the weights of jobs, we cannot make the assumption that the weights are rounded a-priori. Therefore the rounding of the weights is a part of the algorithm.

Let $\mathscr{W}_A$ be defined for $A \in \mathcal{A}_1$ as in Section III, and let $\mathscr{W}'_A$ be defined for $A \in \mathcal{A}_2$ as in Section V.

Definition VII.1. For a bin $A \in \mathcal{A}$ and a time t, we define the score of a bin A to be:

$$\tilde{\mathscr{W}}_A(t) = \begin{cases} \mathscr{W}_A(t) & A \in \mathcal{A}_1 \\ \mathscr{W}'_A(t) & A \in \mathcal{A}_2 \\ w(A(t)) & A \in \mathcal{A}_3 \end{cases}$$

As in previous sections, we denote by $\text{top}_A(t)$ the maximal job in A at time t with respect to the ordering in A.

Theorem VII.2. *Algorithm 3 is* $O(\min(\log(W, P, D)))$ *competitive.*

Proof: In the full version of the paper on arXiv. ■

References

[1] S. Anand, Naveen Garg, and Amit Kumar. *Resource Augmentation for Weighted Flow-time explained by Dual Fitting*, pages 1228–1241.

[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.

[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.

[4] Nikhil Bansal. Minimizing flow time on a constant number of machines with preemption. *Oper. Res. Lett.*, 33(3):267–273, May 2005.

[5] Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 1238–1244, 2009.

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

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

[8] Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 679–684, 2009.

[9] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 363–372, 2004.

[10] Chandra Chekuri and Sanjeev Khanna. Approximation schemes for preemptive weighted flow time. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 297–305, New York, NY, USA, 2002. ACM.

[11] 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.

[12] Naveen Garg and Amit Kumar. Better algorithms for minimizing average flow-time on related machines. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, pages 181–190, 2006.

[13] Naveen Garg and Amit Kumar. Minimizing average flow time on related machines. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 730–738, 2006.

[14] 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.

[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.

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