Theoretical Computer Science 938 (2022) 24–38

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

# Online scheduling of parallelizable jobs in the directed acyclic graphs and speed-up curves models

Benjamin Moseley [a], Ruilong Zhang [b,*,1], Shanjiawen Zhao [c]

[a] Tepper School of Business, Carnegie Mellon University, United States of America
[b] Department of Computer Science, City University of Hong Kong, Hong Kong
[c] Mellon College of Science, Carnegie Mellon University, United States of America

ARTICLE INFO

*Article history:*
Received 17 February 2022
Received in revised form 20 September 2022
Accepted 7 October 2022
Available online 13 October 2022
Communicated by V. Bonifaci

*Keywords:*
Scheduling
DAG jobs
Parallelizable jobs
Online algorithms
$\ell_k$-norm of flow time
Competitive analysis

ABSTRACT

This paper considers scheduling jobs online on $m$ identical machines such that the jobs can be parallelized across the machines. Two models of parallelizability are considered, one is the speed-up curves model, and the other is the directed-acyclic-graph (DAG) model. For both models, the objectives considered are the average, maximum, and $\ell_k$-norms of flow time for $k \geq 1$.

We establish an $\Omega(m)$ lower bound on the competitive ratio of any algorithm for optimizing average flow time in both models without resource augmentation. With resource augmentation, we give a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^{2k+1}})$-competitive algorithm in the DAG model for the $\ell_k$-norms of flow time. This essentially matches the best-known result in the speed-up curve model for the $\ell_k$-norms of flow time. Finally, we show an $O(1)$-competitive algorithm for minimizing the maximum flow time in the speed-up curves model.



## 1. Introduction

This paper considers scheduling processors in multi-programmed systems where $n$ jobs arrive over time online that need to be completed. The system consists of multiple processors, and jobs can be parallelized across the processors. The system scheduler needs to decide how to allocate the processors to the jobs to optimize the quality of service delivered by the system. Throughout this paper, we will assume the $m$ processors are identical and that preemption and migration are allowed. The paper considers an online setting where a job is unknown to the system until it arrives.

There are two popular models that have been considered in scheduling theory to capture the case where jobs can be parallelized across the processors. This paper investigates both of these models and the complexity of the resulting scheduling problems in the models.

**Speed-up Curves Model:** In the speed-up curves model, each job $j$ has a corresponding speed-up function $\Gamma_j : \mathbb{R}^+ \to \mathbb{R}^+$. The value of $\Gamma_j(m')$ is the rate job $j$ is processed when given $m'$ processors. We assume that $\Gamma_j(0) = 0$ and that $\Gamma_j(m')$ is a non-decreasing sublinear function in $m'$. These assumptions are motivated by the following: (1) jobs are not more efficiently

* Corresponding author.
*E-mail addresses:* moseleyb@andrew.cmu.edu (B. Moselely), ruilzhang4-c@my.cityu.edu.hk (R. Zhang), shanjiaz@andrew.cmu.edu (S. Zhao).

[1] This work was done when the author visited Carnegie Mellon University.

https://doi.org/10.1016/j.tcs.2022.10.005
0304-3975/

processed when given additional processors, and (2) the rate of processing does not decrease when a job is given additional resources. See Section 3.1 for details.

**Directed Acyclic Graph (DAG) Model:** In this model, each job $j$ is represented as a directed acyclic graph. Each node in the graph is referred to as a **task** and has a unit processing time. A task can be run once all its predecessors have been completed. Parallelism, in this case, is captured by the fact that multiple tasks can be run on different processors at each point in time. A job is the entire DAG and, as such, a job is complete *when every node in the DAG completes*. See Section 3.2 for details.

**Quality of Service Objectives:** This paper considers schedulers that optimize objectives based on the **flow time** of a job. The flow time is the amount of time a job spends in the system before it is completed. Let $D_j$ be the time that job $j$ is completed, and let $r_j$ be the time job $j$ was **released** (e.g., arrived). The flow time of job $j$ is $D_j - r_j$. Naturally, the client submitting a job would like its flow time minimized. This paper will consider minimizing the $\ell_k$-norms of the flow time $\sqrt[k]{\sum_j (D_j - r_j)^k}$. We will be particularly interested in $k \in \{1, 2, \infty\}$. The $\ell_1$-norm is simply the average flow time, $\ell_\infty$ is the maximum flow time, and $\ell_2$-norm is the variance in the flow times.

### 1.1. Problems considered

These two scheduling models have been of interest recently. We will break down prior work depending on the objective considered.

Consider the *average flow time objective* (i.e., $\ell_1$-norm). For both the DAG [2,1] and speed-up curves [10] models, a $(1+\epsilon)$-speed $\frac{1}{\epsilon^{O(1)}}$-competitive algorithm is known. An algorithm is said to be $s$-speed $c$-competitive if the algorithm is $c$-competitive when running on processors a factor $s$ faster than the optimal solution. This is known as a *resource augmentation* analysis.

It has been of interest to understand the complexity without resource augmentation for average flow time. It is known that for sequential non-parallelizable jobs, the multiprocessor variant of Shortest-Remaining-Processing-Time (SRPT) is $O(\min\{\log P, \log n/m\})$ competitive. Here $P$ is the ratio of maximum to minimum job size. Ideally, similar results could be shown for both parallelism models. Towards this direction, Im et al. [14] showed an $O(\log P)$-competitive algorithm for the speed-up curves setting assuming $\Gamma_j(m') = m'^{\alpha_j}$ when $m' \geq 1$ for some $0 \leq \alpha_j < 1$ and $\Gamma_j(m') = m'$ when $0 \leq m' \leq 1$. The idea is that a job is fully parallel up to one processor. After that, it achieves a speed-up specified by a polynomial function.

**Question 1.** Is there a polylogarithmic competitive algorithm for average flow time in the DAG model? Can the result above be extended to all speed-up curves?

The result of Im et al. in [14] suggests that the answer is likely yes to the above question. Their assumption on the speed-up curves is not too strong, and the result seems likely to extend to all speed-up curves. Moreover, the DAG model usually has similar complexity, and it seems natural to suggest a similar result should exist in the DAG setting.

Next, we discuss the *maximum flow time objective* (i.e., $\ell_\infty$-norm). For this objective, Pruhs, Robert, and Schabanel [20] show that there is a $(1+\epsilon)$-speed $O(\log n)$-competitive algorithm in the speed-up curve setting. Moreover, a lower bound was given, showing that there is no $s$-speed $o(\log n)$-competitive algorithm for any constant $s$, establishing the tightness of the result. Critically, both the upper and lower bounds assume that the algorithm does not know the speed-up curves.

In the DAG model, Agrawal et al. [2] gave a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm that is a variant of the First-In-First-Out (FIFO) algorithm for any $\epsilon > 0$.

**Question 2.** Is there an $O(1)$-competitive algorithm for maximum flow time in the speed-up curve model if the algorithm knows the speed-up functions? Is there an $O(1)$-competitive algorithm in the DAG model for maximum flow time without resource augmentation?

Finally, we discuss the *$\ell_k$-norms of flow time objective* for $1 < k < \infty$. For this objective, Edmonds et al. [9] gave a $(1+\epsilon)$-speed $O(\frac{k}{\epsilon^{2k+1}})$-competitive algorithm in the Speed-up Curves model. No current results are known for the DAG model. It is known that there are strong lower bounds on the competitive ratio of any online algorithm for this problem, even for sequential jobs. Hence, it is necessary to resort to resource augmentation.

**Question 3.** What is the complexity of scheduling to minimize the $\ell_k$ norms of flow time in the DAG model for $1 < k < \infty$?

### 1.2. Our results

This paper focuses on understanding the questions presented. Consider Question 1. For this problem, prior work could be interpreted to suggest that a polylogarithmic competitive algorithm for average flow time is likely to exist for both the

speed-up curves and the DAG scheduling. In fact, we show this is not the case, and there is a strong lower bound! See Section 6 for details.

**Theorem 1.** *Any online algorithm has a competitive ratio at least $\Omega(m)$ for average flow time in the speed-up curves model and the DAG model.*

The instance is not too complicated but illustrates challenges faced in the parallel scheduling environment. Recall that [14] Im et al. showed an $O(\log P)$-competitive algorithm for the speed-up curves setting assuming $\Gamma_j(m') = m'^{\alpha_j}$ when $m' \geq 1$ and $\Gamma_j(m') = m'$ otherwise. Here $0 < \alpha_j < 1$ is known to the algorithm. While this seems to be not a strong assumption, it indeed is what enables prior work to break through the lower bound. Critical is assuming that $\alpha_j \neq 1$ and no job is fully parallelizable.

With this strong lower bound in place, we consider Question 3. We show the answer in the affirmative. See Section 4 for details.

**Theorem 2.** *There exists a $(1 + 10\epsilon)$-speed $O(\frac{1}{\epsilon^{2k}+1})$-competitive algorithm for minimizing the $\ell_k$-norms of flow time in the DAG model for any $\epsilon > 0$ and $1 \leq k < \infty$.*

Finally, we consider Question 2 Recall that [20] showed a lower bound, stating that there is no $s$-speed $o(\log n)$-competitive algorithm for any constant $s$ for maximum flow time in the speed-up curves setting. This lower bound required the algorithm to be oblivious to the speed-up functions.

Assuming the algorithm can use the speed-up functions to make scheduling decisions, we show the following for the speed-up curves model. This breaks through the lower bound. Notice the algorithm does not use resource augmentation. See Section 5 for details.

**Theorem 3.** *There exists a $(16 + \epsilon)$-competitive algorithm for minimizing the maximum flow time in the speed-up curves model.*

This paper does not resolve whether there is an $O(1)$-competitive algorithm for maximum flow time in the DAG setting. This is an intriguing open problem.

## 2. Other related work

Besides the work discussed above, additional work has been done offline and in related models. Two related models for parallelizable jobs are the modable and malleable task models. See [8] for an survey.

The speed-up curves setting is equivalent to the malleable task model. Typically, prior work has referred to the model as malleable tasks when the problem is offline, and the speed-up curves model is used for the online setting. While often true, this is not always the case. The modable tasks model refers to the same problem, except it is more restrictive. In particular, once a number of processors are assigned to a job, they must process the job to completion non-preemptively, and no new processors can be assigned to the job. In some cases, the jobs are assumed to require a precise number of processors to be processed, and this is specified to the scheduler.

The malleable task model has been studied extensively. Over thirty years ago, it was shown that minimizing the makespan offline is strongly NP-Hard [7]. A 2-approximation was shown in [21]. An improved $(\frac{3}{2} + \epsilon)$-approximation algorithm is given by Mounie, Rapine and Trystram [19]. Jansen and Thöle gave a polynomial time approximation scheme when the number of jobs is polynomial in the number of machines [16]. If all speed-up curves are linear, then Drozdowski and Kubiak showed a polynomial time algorithm exists [6].

Approximation algorithms have additionally been given in more general models. Fotakis, Matuschke and Papadigenopoulos consider the setting where machines are not identical but unrelated [11]. Makarychev and Panigrahi [18] and Janesen and Zhang [17] consider malleable tasks that additionally have precedence constraints.

Additional work has been done in the modable task setting, for example [15,5,4,22].

The DAG model has also been extensively considered due to its wide applications in parallel languages and libraries. For the DAG model to minimize the average flow time, Agrawal et al. [2] gave a $(1 + \epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm, which is called Late Arrival Processor Sharing (LAPS). The LAPS algorithm was proposed by Edmonds et al. [10]. Intuitively, the LAPS algorithm shares the processing equally among the latest arriving constant fraction of the jobs. Since the LAPS algorithm is hard to implement, Agrawal et al. [2] also show that the Shortest Job First algorithm (SJF) is a $(2 + \epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm. Intuitively, the SJF algorithm will always schedule the jobs with the least initial work. Beyond the $\ell_k$-norm of the flow time, Agrawal et al. [3] also consider the throughput maximization objective. In this model, each job $j_i$ is associated with a profit function $p_i(t)$, which indicates the profit obtained for finishing job $j_i$ at time $t$. The goal is to find a schedule that maximizes the total obtained profits.

## 3. Problem formulations

### 3.1. Speed-up curves model

In the speed-up curves model, there is a set of jobs $J$ and $m$ identical processors. Job $j$ has an arrival time $r_j$ and a length $p_j$. The actual processing time of job $j$ depends on the number of processors assigned to it. Formally, each job $j$ is associated with a speed-up function $\Gamma_j : \mathbb{R}^+ \to \mathbb{R}^+$. The value of $\Gamma_j(m')$ is the rate job $j$ is processed when given $m'$ processors. Let $m_j(t)$ denote the number of processors assigned to job $j$ at time $t$. Then, the completion time $D_j$ of job $j$ is the time such that $p_j = \int_{t=0}^{D_j} \Gamma_j(m_j(t))$. Let $D_j - r_j$ be the flow time of job $j$. In the online setting, the jobs arrive online. When a job arrives, its speed-up function and length will be known by the algorithm. Preemption and migration are allowed. We assume that all speed-up functions are concave and $\Gamma_j(0) = 0$ for all $j \in J$. Our goal is to minimize the $\ell_\infty$-norm of flow time, i.e., $\max_{j \in J}\{D_j - r_j\}$.

### 3.2. Directed acyclic graph model

In the Directed Acyclic Graph Model, there is a set of jobs $J$. Each job $j$ contains a set of nodes called *tasks*. Each task has a unit processing time, and there are precedence constraints between the tasks. Every job $j \in J$ can be represented as a directed acyclic graph. A task can be processed only if all its predecessor tasks are completed. We say a task is *ready* if all the predecessors of the task had been completed. That is, it can be feasibly scheduled. Each job $j$ has an arrival time $r_j$. A job is completed only if all its tasks are finished.

In the online setting, the algorithms considered in this paper know the ready tasks for a released job at every point in time but do not know the DAG structure. Preemption and migration are allowed. Each processor can only process at most one task at a time. Let $D_i$ be the completion time of job $j$. The goal is to minimize the $\ell_k$-norm of flow time, i.e., $\sqrt[k]{\sum_{j \in J}(D_j - r_j)^k}$. In the resource augmentation model, each processor is given speed $s > 1$ for the online algorithm and is compared to an optimal given speed 1 processors.

Let $W_j$ be the total processing time of the tasks in job $i$'s DAG. Let $C_j$ be the critical-path length of job $j$, where the critical-path is defined as the longest path (the sum of the processing time of tasks on the path) in job $j$'s DAG. Observe that $\max\{\frac{W_j}{m}, C_j\}$ is a lower bound of the processing time of job $j$ in any schedule.

## 4. Upper bound for the DAG model

In this section, we give an upper bound for the DAG model with resource augmentation. We assume that the processors in the algorithm have an extra $10\epsilon$ speed than the optimal solution. We used a potential function analysis framework (see [13] for a detailed overview of the technique). We define a potential function $\Phi(t)$.

To upper bound the competitive ratio of the algorithm, one needs to show the following four conditions. These conditions assume $\Phi(t)$ is continuous except at times when jobs arrive or are completed by the algorithm or optimal schedule.

Let $\mathrm{OPT}(t)$ and $\mathrm{ALG}(t)$ be the total cost in the objective accumulated by a fixed optimal solution and the algorithm at time $t$, respectively. We let $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{ALG}(t)$ and $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$ denote the instantaneous increase in the algorithms cost.

1. Boundary condition: $\Phi(0) = \Phi(\infty) = 0$, that is, the potential function is equal to 0 before any job is released and after all jobs are finished.
2. Arrival condition: the total discontinuous increase of the potential function can be bounded by $\alpha \cdot \mathrm{OPT}$ over the release of all jobs.
3. Completion condition: the total discontinuous increase of the potential function can be bounded by $\beta \cdot \mathrm{OPT}$ over all the completions of jobs by OPT or ALG.
4. Running condition: at any time $t$ when no job arrives or is finished, it is the case that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{ALG}(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq \gamma \cdot \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

Integrating these running conditions over time when the potential is continuous, and summing over the discontinuous changes in the potential, we obtain that the algorithm ALG is $(\alpha + \beta + \gamma)$-competitive. See [13] for more details.

The remainder of this section is organized as follows. In Section 4.1, we introduce the Weighted Latest Arrival Processor Sharing algorithm. Then, we define our potential function in Section 4.2 and give some intuition in Section 4.3. Finally, we give the analysis in Section 4.4.

### 4.1. WLAPS algorithm

The algorithm we use is called Weighted Latest Arrival Processor Sharing (WLAPS for short), which is proposed by [12]. We consider the resource-augmented setting. That is, the algorithm is given $m$ machines with $(1 + 10\epsilon)$-speed, while the optimal solution only has $m$ machines with 1 speed. Here $0 < \epsilon < \frac{1}{10}$ is a fixed constant. To formally describe the algorithm, we introduce some notation. Let $N_a(t)$ be the set of released jobs that are not yet completed. For each job $i$, let $w_i(t) = k(t - r_i)^{k-1}$ be its weight at the time $t$. Let $N'_a(t) \subseteq N_a(t)$ be the set of jobs in $N_a(t)$ with the latest arrival times

such that $\sum_{i\in N'_a(t)} w_i(t) = \beta \sum_{i\in N_a(t)} w_i(t)$, where $\beta$ is the parameter chosen by the algorithm. In this work, we set $\beta = \epsilon^k$. At each time point $t$, the algorithm WLAPS will share its machines among the jobs in $N'_a(t)$ according to the proportion of their weights. Formally, for an arbitrary job $j \in N'_a(t)$, the algorithm will assign $m \cdot \frac{w_i(t)}{\beta \sum_{i\in N_a(t)} w_i(t)}$ machines to the job.

Note that the WLAPS algorithm requires that there always exists a set of jobs whose weights are exactly $\beta \sum_{i\in N_a(t)} w_i(t)$. Otherwise, the WLAPS algorithm will process the set of minimal latest arriving jobs whose weights exceed $\beta \sum_{i\in N_a(t)} w_i(t)$. Let $N'_a(t)$ be such a job set. Let $j$ be the job in $N'_a(t)$ with the earliest arrival time. Then, for every job $i$ in $N'_a(t) \setminus \{j\}$, the number of assigned machines is the same as before except for job $j$. For job $j$, the number of assigned machines is proportional to its weight which overlaps the $\beta$ fraction of the total weight, i.e., $m \cdot (1 - \frac{\sum_{i\in N'_a(t)\setminus\{j\}} w_i(t)}{\beta \sum_{i\in N_a(t)} w_i(t)})$. In the remainder of this work, we assume that a set of the latest arriving jobs whose total weight is exactly $\beta \sum_{i\in N_a(t)} w_i(t)$ always exists. The assumption has no impact on the analysis but simplifies the proof.

### 4.2. Potential function

We re-index the jobs by their arrival time. Break ties arbitrarily if jobs have the same arrival time. Recall that $r_i$ is the release time of job $i$. We define $N_a(t)$ as the set of released yet unfinished jobs in the algorithm ALG. Let $S_i^A(t)$ and $S_i^O(t)$ be the amount of work remaining for job $i$ in ALG and OPT at time point $t$, respectively. Then we define $Z_i(t)$ be the lag of job $i$ at time point $t$, i.e., $Z_i(t) := \max\left\{S_i^A(t) - S_i^O(t), 0\right\}$. Let $w_i(t) := k(t - r_i)^{k-1}$ be the weight of job $i$ at time point $t$. It is worth noticing the following well-known observation:

$$\text{ALG} = \int_0^\infty \left(\frac{\mathrm{d}}{\mathrm{dt}}\text{ALG}(t)\right)\mathrm{dt} = \sum_{i\in[n]}(D_i - r_i) = \int_0^\infty \sum_{i\in N_a(t)} w_i(t)\mathrm{dt}, \tag{1}$$

where $D_i$ is the completion time of job $i$ in ALG's schedule. Let $c_i(t)$ be the remaining length of the critical path of job $i$ in ALG at time $t$. Adopting ideas from [9], we define the potential function as follows:

$$\Phi(t) := \sum_{i\in N_a(t)} \left(t - r_i + \frac{1}{m\epsilon} \sum_{r_j \geq r_i, j\in N_a(t)} Z_j(t)\right)^k + \left(\frac{2}{\epsilon}\right)^{2k+1} \sum_{i\in N_a(t)} w_i(t)c_i(t).$$

For the sake of analysis, let $\Phi_1(t)$ and $\Phi_2(t)$ be the first term and the second term of $\Phi(t)$, respectively.

### 4.3. Intuition

In this section, we briefly introduce the intuition behind the potential function $\Phi(t) = \Phi_1(t) + \Phi_2(t)$. To show the competitive ratio, we need to show that at each time $t$, the total age$^{k-1}$ of the alive jobs is at most some constant times the corresponding value for those under the optimal schedule. Fortunately, we can utilize the potential function to make an amortized comparison between the algorithm and the optimal schedule. In the potential function analysis framework, the potential function is considered a bank. At any time $t$, the algorithm can store its cost when the algorithm's cost is smaller than the optimal schedule's cost. And the algorithm can withdraw these reserves when its cost is higher than the optimal solution.

Our potential function contains two terms: $\Phi_1(t)$ and $\Phi_2(t)$. The sum of the two is an estimate of the algorithm's remaining cost for the remaining incomplete jobs assuming no more jobs arrive. Intuitively, the derivation of $\Phi_1(t)$ approximates the remaining cost jobs will pay in the ALG's schedule during times jobs are in their parallel phases. That is, if the jobs are fully parallelizable, this estimates the difference in the algorithm's remaining cost compared to the optimal remaining cost. Notice that $(t - r_i)$ is the flow time of job $i$ at time $t$. The term $\frac{1}{m\epsilon}\sum_{r_j\geq r_i, j\in N_a(t)} Z_j(t)$ is an estimate of the remaining waiting time of job $i$ in the WLAPS algorithm, assuming no more jobs arrive.

Recall that WLAPS will first process jobs in $N_a(t)$ with the latest arrival times. This estimate assumes the jobs are completed in the reverse arrival order, each job $j$'s size is $Z_j(t)$, and jobs can be fully parallelized on $m$ machines. Of course, these assumptions may not hold, but this is an estimate. An estimate of $i$'s completion time is $t - r_i + \frac{1}{m\epsilon}\sum_{r_j\geq r_i, j\in N_a(t)} Z_j(t)$. The $\epsilon$ is added for technical reasons.

Of course, the jobs are not fully parallelizable. This is where the $\Phi_2(t)$ is needed. Intuitively, the $\Phi_2(t)$ approximates the remaining cost that jobs will pay in the ALG's schedule due to not being fully parallelizable since the length of the critical path lower bounds the cost of any algorithm for the time it takes to complete a job.

### 4.4. Analysis

In the following, we will find suitable parameters $\alpha, \beta, \gamma$ such that the algorithm WLAPS satisfies the above four conditions. We will begin with the easy conditions, i.e., boundary, arrival, and completion condition. We will show that these

three conditions will not increase the potential function. Basically, these properties follow directly from the definition of the potential function. For the running condition, we bound the increase of the potential function in three scenarios: (1) Time Elapse: the change in $\Phi(t)$ due to the change in time; (2) OPT's Processing: the change in $\Phi(t)$ due to the processing of the optimal solution; (3) WLAPS's Processing: the change in $\Phi(t)$ due to the processing of the algorithm. Finally, we integrate all the conditions and obtain the competitive ratio.

*Boundary condition* At time 0 and $\infty$, we have no jobs; therefore, set $N_a(t)$ is empty, which implies $\Phi(0) = \Phi(\infty) = 0$.

*Completion condition* Completing job $j$ by OPT does not affect the critical path and the $Z_j$ term. When WLAPS completes a job $j$, it is removed immediately from $N_a(t)$, so both $Z_j$ and $C_j$ are zero. Therefore, the completion of jobs in both OPT and WLAPS does not increase the value of the potential function.

*Arrival condition* Since $t - r_i$ and $Z_i$ are both zero, the arrival of jobs does not increase the $\Phi_1(t)$. For the $\Phi_2(t)$, since $w_i(t) = k(t - r_i)^{k-1} = 0$ when job $i$ just arrived, $\Phi_2(t)$ does not increase when new jobs arrive either.

*Time elapse* Note that, in the potential function, both $t$ and $Z_j(t)$ change over time. Here we only bound the change of $t$ and analyze the change of $Z_j(t)$ later together with OPT and ALG's processing. Taking the derivatives, the change in our potential function due to time elapse for the first term $\Phi_1(t)$ is

$$\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) = \sum_{i\in N_a(t)} k\left(t - r_i + \frac{1}{m\epsilon}\sum_{r_j\geq r_i,\, j\in N_a(t)} Z_j(t)\right)^{k-1}. \tag{2}$$

For notation convenience, we define

$$W_i(t) := k\left(t - r_i + \frac{1}{m\epsilon}\sum_{r_j\geq r_i,\, j\in N_a(t)} Z_j(t)\right)^{k-1}.$$

Note that $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) = \sum_{i\in N_a(t)} W_i(t)$. For $\Phi_2(t)$, the increment is

$$\frac{\mathrm{d}}{\mathrm{dt}}\Phi_2(t) = \left(\frac{2}{\epsilon}\right)^{2k+1}\sum_{i\in N_a(t)} k(k-1)(t-r_i)^{k-2}c_i(t).$$

Let $C_i$ be the length of the critical path of job $i$. To bound $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_2(t)$, we partition the job set $N_a(t)$ into two subsets: old job set $\mathcal{O}(t)$ and young job set $\mathcal{Y}(t)$, where jobs in $\mathcal{O}(t)$ have been alive for more than $k(\frac{2}{\epsilon})^{2k+1}C_i$; otherwise, in $\mathcal{Y}(t)$, i.e., $\mathcal{O}(t) = \{\, i \in N_a(t) \mid (t - r_i) \geq k(\frac{2}{\epsilon})^{2k+1}C_i \,\}$ and $\mathcal{Y}(t) = N_a(t) \setminus \mathcal{O}(t)$. Then the change of potential in $\Phi_2(t)$ due to old jobs can be bounded by:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{dt}}\Phi_2(t) &= \left(\frac{2}{\epsilon}\right)^{2k+1}\sum_{i\in\mathcal{O}(t)} k(k-1)(t-r_i)^{k-2}c_i(t)\\ &\leq \sum_{i\in\mathcal{O}(t)} k(t-r_i)^{k-1}\\ &\leq \sum_{i\in N_a(t)} w_i(t)\\ &\leq \sum_{i\in N_a(t)} W_i(t).\end{aligned} \tag{3}$$

In total, the amount of increase in $\Phi(t)$ by old jobs due to the change in time can be bounded by $2\sum_{i\in N_a} W_i(t)$. On the other hand, for the total change of potential in $\Phi_2(t)$ due to young jobs, we will charge it later to the cost of the optimal solution. We first give Lemma 1.

**Lemma 1.** $\left(\frac{2}{\epsilon}\right)^{2k+1}\int_0^\infty \sum_{i\in\mathcal{Y}(t)} k(k-1)(t-r_i)^{k-2}c_i(t)\mathrm{dt} \leq k^k(\frac{2}{\epsilon})^{2k(2k+1)}\mathrm{OPT}$.

**Proof.** We have the following inequalities:

$$\left(\frac{2}{\epsilon}\right)^{2k+1}\int_0^\infty \sum_{i\in\mathcal{Y}(t)} k(k-1)(t-r_i)^{k-2}c_i(t)\mathrm{dt}$$

$$\leq \left(\frac{2}{\epsilon}\right)^{2k+1} \int_0^\infty \sum_{i\in\mathcal{Y}(t)} k(k-1)\left(k\left(\frac{2}{\epsilon}\right)^{2k+1} C_i\right)^{k-2} C_i \mathrm{dt}$$

$$= \left(\frac{2}{\epsilon}\right)^{2(k-2)(2k+1)} k^{k-1}(k-1)\int_0^\infty \sum_{i\in\mathcal{Y}(t)} C_i^{k-1}\mathrm{dt}$$

$$\leq k^k \left(\frac{2}{\epsilon}\right)^{2k(2k+1)} \mathrm{OPT},$$

where the first inequality is due to the definition of the job set $\mathcal{Y}(t)$ and $c_i(t) \leq C_i$ holds for all jobs, and the last inequality is due to $\int_0^\infty \sum_{i\in[n]} C_i^{k-1} \leq \mathrm{OPT}$. □

OPT*'s processing* Since OPT's processing does not change the critical path in the schedule returned by the algorithm for any jobs, we only have to worry about the first term of our potential function, i.e., $\Phi_2(t)$ remains the same during the OPT's processing. Observe that the worst-case scenario is when the optimal solution processes the job with the latest arrival time. Let this be job $k$. Since $Z_k = S_k^A - S_k^O$ and optimal solution has speed 1, processing job $k$ at speed 1 increases $Z_k$ by $m$. Therefore, we have

$$\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \leq \frac{1}{\epsilon}\sum_{i\in N_a(t)} k\left(t - r_i + \frac{1}{m\epsilon}\sum_{r_j\geq r_i, j\in N_a(t)} Z_j(t)\right)^{k-1} = \frac{1}{\epsilon}\sum_{i\in N_a(t)} W_i(t). \tag{4}$$

ALG*'s processing* We first bound the change of $\Phi_1(t)$ due to the algorithm's processing. For the change of $\Phi_2(t)$, we will directly charge it to the OPT. Recall that $N'_a(t)$ is the job set that WLAPS chooses to process at time $t$. Then, we have the following lemma.

**Lemma 2.** *The change of the $\Phi_1(t)$ due to* WLAPS*'s processing can be bounded by the following term:*

$$\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) \leq -\frac{1}{m\epsilon}\left(\sum_{j\in N'_a(t)\setminus N_o(t)} m\frac{w_j(t)}{\beta w(t)}\right)\cdot\left(\sum_{i\in N_a(t)\setminus N'_a(t)} W_i(t)\right),$$

*where $w(t) = \sum_{i\in N_a(t)} w_i(t)$.*

**Proof.** Consider any job $j \in N'_a(t) \setminus N_o(t)$, $Z_j(t)$ is the remaining work to be processed by WLAPS since OPT would have completed the job $j$. Thus, $Z_j(t)$ will be decreased by $m\frac{w_i(t)}{\beta w(t)}$ by definition of the algorithm. Since all jobs in $N'_a(t)$ have a larger arrival time than all jobs in $N_a(t) \setminus N'_a(t)$, $Z_j(t)$ will decrease by the above term for all jobs in $N_a(t) \setminus N'_a(t)$. Thus, Lemma 2 directly follows. □

Observe that the change of the potential function due to OPT's processing and the change in time can be bounded by some factor multiplying $\sum_{i\in N_a(t)} W_i(t)$. In order to integrate all the conditions, we need to obtain another upper bound of $\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t)$ that is related to $\sum_{i\in N_a(t)} W_i(t)$. This can be captured by Lemma 3.

**Lemma 3.** *For any subset of jobs $N''_a(t)$ such that $\sum_{i\in N''_a(t)} w_i = \beta\sum_{i\in N_a(t)} w_i(t)$, we have:*

$$\sum_{i\in N_a(t)\setminus N''_a(t)} W_i(t) \geq \left(1 - \beta(1+\frac{1}{\epsilon})^{k-1}\right)\sum_{i\in N_a(t)} W_i(t).$$

**Proof.** We have the following inequalities:

$$\sum_{i\in N_a(t)\setminus N''_a(t)} W_i(t) = \sum_{i\in N_a(t)} W_i(t) - \sum_{i\in N''_a(t)} W_i(t)$$

$$\geq \sum_{i\in N_a(t)} W_i(t) - (1+\frac{1}{\epsilon})^{k-1}\sum_{i\in N''_a(t)} w_i(t)$$

$$= \sum_{i\in N_a(t)} W_i(t) - \beta(1+\frac{1}{\epsilon})^{k-1}\sum_{i\in N_a(t)} w_i(t)$$

$$\geq \left(1-\beta(1+\frac{1}{\epsilon})^{k-1}\right)\sum_{i\in N_a(t)} W_i(t). \quad \square$$

Combining Lemma 2 and Lemma 3, we have:

$$\frac{\mathrm{d}}{\mathrm{dt}}\Phi_1(t) \leq -\frac{1}{m\epsilon}\cdot\left(1-\beta(1+\frac{1}{\epsilon})^{k-1}\right)\cdot\left(\sum_{j\in N'_a(t)\setminus N_o(t)} m\frac{w_j(t)}{\beta w(t)}\right)\cdot\sum_{i\in N_a(t)} W_i(t). \tag{5}$$

*Integrating the conditions* To obtain the competitive ratio, we will integrate over the above changes in the potential. Recall that in Lemma 1, we have bounded the total change due to time elapse for the young jobs. Let $\Phi'(t)$ be the total increase in $\Phi(t)$ after excluding the young jobs. Then, we have the following inequalities:

$$\begin{aligned}
\mathrm{WLAPS} &= \int_0^\infty \left(\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)\right)\mathrm{dt} \\
&= \int_0^\infty \left(\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t)\right)\mathrm{dt} \\
&\leq \int_0^\infty \left(\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)\right)\mathrm{dt} + k^k(\frac{2}{\epsilon})^{2k(2k+1)}\cdot\mathrm{OPT}.
\end{aligned} \tag{6}$$

Now, it remains to bound $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)$ by OPT. After integrating Equation (4), Equation (2), Equation (3) and Equation (5), we have:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t) \\
&= (2+\epsilon)\sum_{i\in N_a(t)} W_i(t) - \frac{1}{\epsilon}\cdot\left(1-\beta(1+\frac{1}{\epsilon})^{k-1}\right)\cdot\left(\sum_{j\in N'_a(t)\setminus N_o(t)}\frac{w_j(t)}{\beta w(t)}\right)\cdot\sum_{i\in N_a(t)} W_i(t).
\end{aligned} \tag{7}$$

In the following, we mainly show Lemma 4.

**Lemma 4.** $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t) \leq (\frac{1}{\epsilon})^{k+1}(2+\frac{1}{\epsilon})(1+\frac{1}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

Combining Equation (6) and Lemma 4, Theorem 2 directly follows.

**Proof of Theorem 2.** Combining Equation (6) and Lemma 4, we have the competitive ratio of WLAPS as follows:

$$\begin{aligned}
\mathrm{WLAPS} &\leq \int_0^\infty \left(\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)\right)\mathrm{dt} + k^k(\frac{2}{\epsilon})^{2k(2k+1)}\cdot\mathrm{OPT} \\
&\leq \left((\frac{1}{\epsilon})^{k+1}(2+\frac{1}{\epsilon})(1+\frac{1}{\epsilon})^{k-1} + k^k(\frac{2}{\epsilon})^{2k(2k+1)}\right)\cdot\mathrm{OPT}. \quad \square
\end{aligned} \tag{8}$$

Now, it remains to show Lemma 4. Before giving the proof, we first provide the following two lemmas, which help show Lemma 4.

**Lemma 5.** *For any job i,* $\sum_{r_j\geq r_i, r_j\in N_a(t)} Z_i \leq m(t-r_i)$.

**Proof.** Notice that $m(t-r_i)$ is the amount of time job $i$ has been alive and $\sum_{r_j\geq r_i, r_j\in N_a(t)} Z_i$ is the amount of work that WLAPS is lagging compared to OPT for jobs that arrived after job $i$. Since OPT processes at speed 1, the largest amount that WLAPS is lagging can only be $m(t-r_i)$. $\square$

**Lemma 6.** $\sum_{i\in N_a(t)} k\left(t-r_i+\frac{1}{m\epsilon}\sum_{r_j\geq r_i, j\in N_a(t)} Z_j(t)\right)^{k-1} \leq (1+\frac{1}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)$.

**Proof.** We have the following inequalities:

$$\begin{aligned}\sum_{i\in N_a(t)} k\left(t-r_i+\frac{1}{m\epsilon}\sum_{r_j\geq r_i, j\in N_a(t)} Z_j(t)\right)^{k-1} &\leq \sum_{i\in N_a(t)} k\left(t-r_i+\frac{1}{m\epsilon}m(t-r_i)\right)^{k-1}\\ &= \sum_{i\in N_a(t)}(1+\frac{1}{\epsilon})^{k-1}k(t-r_i)^{k-1}\\ &= \sum_{i\in N_a(t)}(1+\frac{1}{\epsilon})^{k-1}w_i\\ &= (1+\frac{1}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t),\end{aligned}$$

where the first equality is due to Lemma 5 and the second equality is due to Equation (1). □

**Proof of Lemma 4.** To bound $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)$ by OPT, we distinguish three cases.

*Case I. There are at least $\epsilon\beta w(t)$ total weight of jobs in* ALG*'s schedule that are also in* OPT. In this case, we have $\sum_{i\in N_o(t)} w_i(t)\geq \epsilon\beta\sum_{i\in N_a(t)} w_i(t)$. We can charge the increase in cost directly to the increase in the optimal cost. Considering the amount of increase caused by OPT's processing and time elapse, we know that the total increase of potential is $(2+\frac{1}{\epsilon})\sum_{i\in N_a(t)} W_i(t)$. Lemma 6 bounds the $\sum_{i\in N_a(t)} W_i(t)$ by $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)$. The condition of the current case provides the relationship between $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)$ and $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t)$.

Thus we have the following inequalities:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t) &\leq (2+\frac{1}{\epsilon})\sum_{i\in N_a} W_i \leq (2+\frac{1}{\epsilon})(1+\frac{1}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)\\ &\leq \frac{(2+\frac{1}{\epsilon})(1+\frac{1}{\epsilon})^{k-1}}{\epsilon\beta}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t),\end{aligned} \tag{9}$$

where the last equality is due to the condition of the current case.

*Case II. There are at least $\epsilon^2 w(t)$ weights of jobs in* ALG *whose critical paths are decreased.* In this case, the decrease of the critical path in $\Phi_2(t)$ can pay for the increase elsewhere. Since the algorithm has speed augmentation for $(1+10\epsilon)$, $\Phi_2(t)$ will be decreased by $\left(\frac{2}{\epsilon}\right)^{2k+1}(1+10\epsilon)\epsilon^2 w(t)$. Now we are ready to bound $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)$ in this case. Observe that $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)\leq(2+\frac{1}{\epsilon})\sum_{i\in N_a} W_i(t)$. Thus, we charge the decrease of the critical path to $(2+\frac{1}{\epsilon})\sum_{i\in N_a} W_i(t)$. Then, we have

$$\begin{aligned}&(2+\frac{1}{\epsilon})\sum_{i\in N_a(t)} W_i(t)-\left(\frac{2}{\epsilon}\right)^{2k+1}(1+10\epsilon)\epsilon^2\sum_{i\in N_a(t)} w_i(t)\\ &\leq (2+\frac{1}{\epsilon})(1+\frac{1}{\epsilon})^{k-1}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)-\left(\frac{2}{\epsilon}\right)^{2k+1}(1+10\epsilon)\epsilon^2\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)\\ &\leq 0,\end{aligned} \tag{10}$$

where the first inequality is due to Lemma 6 and Equation (1). Thus, we have $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)\leq 0$ in Case II.

*Case III. There are less than $\epsilon^2 w(t)$ weights of jobs whose critical paths decrease and less than $\epsilon\beta w(t)$ weights of jobs are also in* OPT. Recall that we need to bound $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{WLAPS}(t)+\frac{\mathrm{d}}{\mathrm{dt}}\Phi'(t)$ in this case. We first bound the second term of Equation (7).

$$\begin{aligned}&-\frac{1}{\epsilon}\cdot\left(1-\beta(1+\frac{1}{\epsilon})^{k-1}\right)\cdot\left(\sum_{j\in N'_a(t)\setminus N_o(t)}\frac{w_j(t)}{\beta w(t)}\right)\cdot\sum_{i\in N_a(t)} W_i(t)\\ &=-\frac{1}{\epsilon\beta w(t)}\left(1-\beta(1+\frac{1}{\epsilon})^{k-1}\right)\cdot\left(\sum_{j\in N'_a(t)\setminus N_o(t)} w_j(t)\right)\cdot\sum_{i\in N_a(t)} W_i(t)\\ &\leq -\frac{1}{\epsilon\beta}(1-\epsilon\beta)\left(1-\beta(1+\frac{1}{\epsilon})^{k-1}\right)\cdot\sum_{i\in N_a(t)} W_i(t),\end{aligned} \tag{11}$$

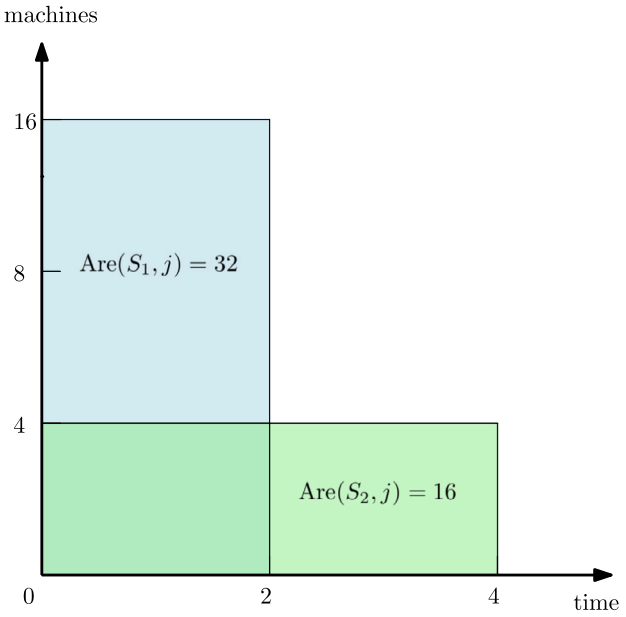


**Fig. 1.** Illustration for Observation 1. The processing time $p_j = 8$ and the speed-up function $\Gamma_j(x) = \sqrt{x}$. In schedule $S_1$, the completion time of job $j$ is 2 and, therefore 16 machines are needed. In schedule $S_2$, the completion time of job $j$ is 4, and therefore only 4 machines are needed. The occupied machine time in $S_1$ is larger than $S_2$.

where the last inequality is due to the condition of the current case. Combining Equation (7) and Equation (11), setting $\beta = \epsilon^k$, we can bound the $\frac{d}{dt}\text{WLAPS}(t) + \frac{d}{dt}\Phi'(t)$ as follows:

$$\begin{aligned}
&\frac{d}{dt}\text{WLAPS}(t) + \frac{d}{dt}\Phi'(t) \\
&= (2+\epsilon)\sum_{i\in N_a(t)} W_i(t) - \frac{1}{\epsilon}\cdot\left(1-\beta(1+\frac{1}{\epsilon})^{k-1}\right)\cdot\left(\sum_{j\in N'_a(t)\setminus N_o(t)}\frac{w_j(t)}{\beta w(t)}\right)\cdot\sum_{i\in N_a(t)} W_i(t) \\
&\le \left(2+\frac{1}{\epsilon}-\frac{1}{\epsilon\beta}(1-\epsilon\beta)\left(1-\beta(1+\frac{1}{\epsilon})^{k-1}\right)\right)\cdot\sum_{i\in N_a(t)} W_i(t) \\
&\le 0.
\end{aligned} \tag{12}$$

Combining Equation (9), Equation (10) and Equation (12), we have:

$$\frac{d}{dt}\text{WLAPS}(t) + \frac{d}{dt}\Phi'(t) \le \frac{(2+\frac{1}{\epsilon})(1+\frac{1}{\epsilon})^{k-1}}{\epsilon\beta}\frac{d}{dt}\text{OPT}(t) = (\frac{1}{\epsilon})^{k+1}(2+\frac{1}{\epsilon})(1+\frac{1}{\epsilon})^{k-1}\frac{d}{dt}\text{OPT}(t). \quad \square$$

## 5. Speed-up curves model

In this section, we give an $O(1)$-competitive algorithm for **maximum flow time** objective in the speed-up curves model. Before introducing the algorithm, we first introduce some notations and give the following simple observation (Observation 1). Consider an arbitrary schedule $S$ and a job set $J'$. Let Are$(S, J')$ be the total machine time occupied by the jobs in $J'$ in the schedule $S$. *The total machine time* is defined to be the summation over machines of the total number of time steps devoted to jobs in $J'$. Let Com$(S, j)$ be the completion time of job $j$ in the schedule $S$. Observation 1 fundamentally relies on the fact that the speed-up function is concave.

**Observation 1.** Consider a job $j$ with release time 0 and processing time $p_j$. Let $S_1$ and $S_2$ be two schedules where $\text{Com}(S_1, j) \le \text{Com}(S_2, j)$. Then, we have $\text{Are}(S_1, j) \ge \text{Are}(S_1, j)$. A simple example can be found in Fig. 1.

Now, we are ready to give the algorithm. Our algorithm is First In First Out (FIFO), and we first assume that we know an upper bound $\kappa$ of the optimum value OPT. The standard doubling algorithm can remove the above assumption (see Observation 2). Consider jobs in FIFO order. When considering $j$, let $m_j$ be the number of machines such that $\frac{p_j}{\Gamma_j(m_j)} = \kappa$. Note that $m_j$ may not be an integer. Let $I(t)$ be the number of idle machines currently at time $t$. Our algorithm assigns

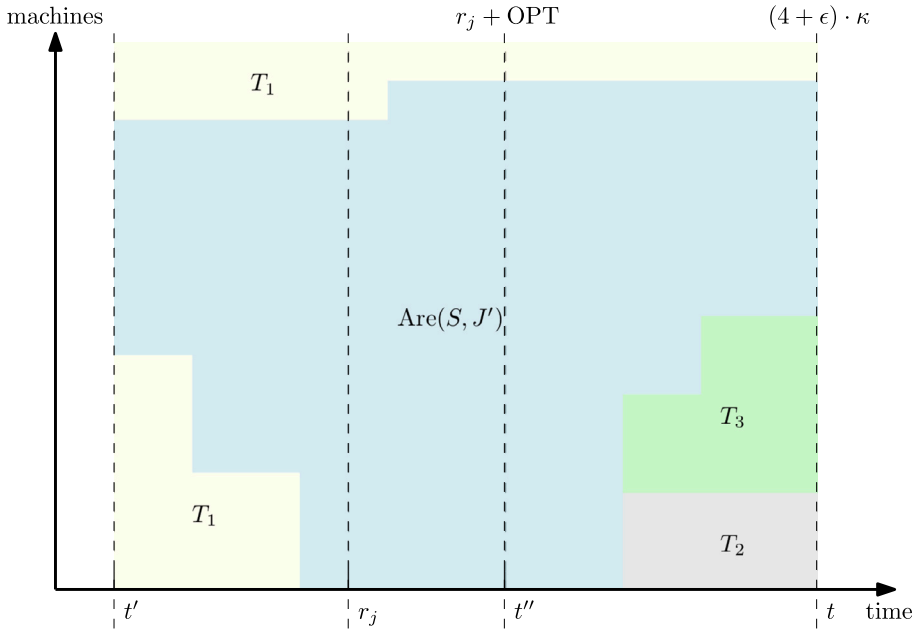


**Fig. 2.** Illustration for the proof of Lemma 7. The yellow area is the total machine time occupied by jobs that arrived before $t'$, i.e., $T_1$. The blue area ($\mathrm{Are}(S, J')$) is the total machine time occupied by the jobs in $J'$. The green area ($T_3$) is the machine time occupied by job $j$, and the gray area ($T_2$) is the idle machine time. The optimal solution has to finish all jobs in $J'$ by $t''$. However, it is impossible if job $j$ cannot be completed by time $t$ due to Observation 1. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$\min\{I(t), m_j\}$ machines to job $j$, decreases $I(t)$, and considers the next job in FIFO order. This continues until all machines are non-idling or there are no more jobs left to consider.

**Lemma 7.** *Let* ALG *be the solution returned by the algorithm. Then, we have* $\mathrm{ALG} \le (4+\epsilon)\cdot\kappa$*, where* $\epsilon > 0$ *can be sufficiently small, and* $\kappa \ge \mathrm{OPT}$ *is an upper bound of the optimum value.*

**Proof.** Let $S$ be the schedule returned by the algorithm. Let $S'$ be the optimal schedule. We only need to show that every job $j$ can be completed by $r_j + (3+\epsilon)\cdot\kappa$. For the sake of contradiction, we assume that there exists a job $j$ such that the job cannot be completed by $r_j + (3+\epsilon)\cdot\kappa$. For convenience, let $t = r_j + (3+\epsilon)\cdot\kappa$. Let $t'$ be the earliest time before $r_j$ such that all machines are occupied by some jobs in schedule $S$. Let $J'$ be the set of jobs that are released between $t'$ and $r_j$ in the schedule $S$, i.e., $J' = \{\, i \in J \mid t' \le r_i \le r_j \,\}$. Note that all jobs in $J'$ will be completed by $r_j + \mathrm{OPT}$ in the optimal schedule $S'$. Note that we do not care about the jobs released after $r_j$ since our algorithm is FIFO. The jobs with a larger release time will never delay job $j$. Observe that if we can show that $\mathrm{Are}(S, J') > (r_j - t')\cdot m + m\cdot\mathrm{OPT}$, then the optimal solution cannot complete $J'$ by $r_j + \mathrm{OPT}$ since $\mathrm{Are}(S', J') \ge \mathrm{Are}(S, J')$ due to Observation 1. Thus, in the following, we mainly show the inequality above by using our assumption: job $j$ cannot be completed by time $t$.

Note that there may exist some jobs that arrived before the time $t'$ but are still alive in the interval $[r_j, t]$. There may also exist some jobs that arrived before the time $t'$ and are completed in the interval $(t', r_j)$. By definition of the algorithm, the total machine time occupied by these jobs is at most $m\cdot\kappa$. Let $T_1$ be the total machine time in $[t', t]$ occupied by these jobs, i.e., $T_1 \le m\cdot\kappa$. Since job $j$ cannot be completed by $t$, the total idle machine time in the interval $[t_j, t]$ is at most $m_j\cdot\kappa \le m\cdot\kappa$ by the algorithm. Let $T_2$ be such machine time, i.e., $T_2 \le m\cdot\kappa$. Note that part of job $j$ may be finished by $t$, and this part is also at most $m_j\cdot\kappa \le m\cdot\kappa$. Let $T_3$ be such machine time, i.e., $T_3 \le m\cdot\kappa$. Thus, the total machine time in the interval $[r_j, t]$ that is occupied by the jobs in $J'$ is at least $(t-t')\cdot m - T_1 - T_2 - T_3 = (t-t')\cdot m - 3m\kappa$, i.e., $\mathrm{Are}(S, J') \ge (t-t')\cdot m - 3m\kappa$. An example can be found in Fig. 2.

Note that $(t-t')\cdot m = (r_j - t')\cdot m + (t - r_j)\cdot m$. Since $t = r_j + (4+\epsilon)\cdot\kappa$, we have $(t-r_j)\cdot m > 4\cdot m\kappa$. Thus, we have $(t-t')\cdot m > (r_j - t')\cdot m + m\cdot\kappa \ge (r_j - t')\cdot m + m\cdot\mathrm{OPT}$. Thus, the optimal solution cannot finish $J'$ by $r_j + \mathrm{OPT}$, which contradicts our assumption. □

**Observation 2.** Given an initial guess $\kappa_0$ of the optimum value OPT and an online algorithm ALG$'$ such that $\mathrm{ALG}' \le \alpha\cdot\kappa$, there is an online algorithm ALG such that $\mathrm{ALG} \le 4\alpha\cdot\mathrm{OPT}$, where $\kappa_0 \in (0, \mathrm{OPT}]$ and $\kappa \ge \mathrm{OPT}$.

**Proof.** Observation 2 is a simple application of the well-known doubling strategy. The doubling algorithm ALG starts from the initial guess $\kappa_0$ of the optimum value. If algorithm ALG$'$ cannot schedule all jobs using the bound $\kappa_i$, then ALG will run ALG$'$ with the new bound $\kappa_{i+1} = 2\kappa_i$. For every computation with the new bound, ALG will ignore all the decisions made

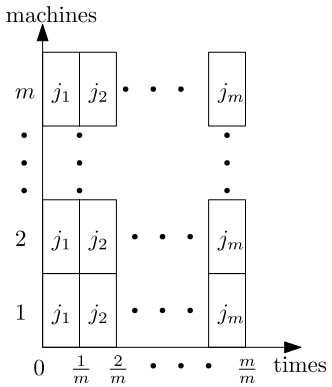


**Fig. 3.** Illustration for Observation 3. If all jobs are fully parallel, the optimal solution will distribute each job equally to $m$ machines in every $\frac{1}{m}$ time points.

so far but keep the cost of the previous decision. One can imagine that the release time of every job will be added by $\kappa_i$ at the beginning of the iteration with bound $\kappa_{i+1}$. Assume that, after $(k-1)$-iteration, ALG finds the first upper bound of OPT, i.e., $\kappa_k \geq \text{OPT}$. Then, we have the following inequality:

$$\text{ALG} \leq \sum_{i=0}^{k} \alpha \cdot \kappa_i = \alpha \cdot (2^{k+1} - 1) \cdot \kappa_0 \leq \alpha \cdot 4 \cdot \kappa_{k-1} \leq \alpha \cdot 4 \cdot \text{OPT},$$

where the second inequality is due to $2^{k-1}\kappa_0 = \kappa_{k-1}$, and the last inequality is due to $\kappa_{k-1} \leq \text{OPT}$. □

By Lemma 7 and Observation 2, Theorem 3 directly follows.

## 6. Lower bound for DAG model

In this section, we give an $\Omega(m)$-lower bound for the DAG model without resource augmentation for the average flow time objective. To show the lower bound, we give a hard instance in the proof of Theorem 1. The hard instance contains two rounds of jobs. The first round of jobs is described in Observation 4 and the second round of jobs is stated in Observation 5. Intuitively, for an algorithm to be $O(m)$-competitive, the algorithm has to schedule the first round's job in a similar manner described in Fig. 4 (Lemma 8). In this case, the adversary will release the second round's jobs. Lemma 9 shows that any $O(m)$-competitive algorithm cannot delay the second round's jobs too much. This will contradict the claim stated in Lemma 8.

Before giving the proofs, we first provide the following simple observation.

**Observation 3.** Let $J$ be a job set consisting of $m$ fully parallel jobs of size 1, where $m$ is the number of machines. All jobs in $J$ are released at time 1. Then, the optimal solution will schedule them in a fully parallel way. An example can be found in Fig. 3. Then, the optimal solution has value $\text{OPT}(J) = \frac{m+1}{2}$.

**Observation 4.** Let $J = J_1 \cup J_2$ be a job set, where $J_1$ is a sequential job set, and $J_2$ is a parallel job set. There are $\frac{m}{2}$ sequential jobs in $J_1 = \{j_1^1, \ldots, j_m^1\}$. All jobs in $J_1$ are released at time point 0 with the size $L$. Let $J_2 = \{j_1^2, \ldots\}$, where there are $\frac{mL}{2}$ parallel jobs. Job $j_i^2$ is released at time point $\frac{i-1}{m}$. All jobs in $J_2$ have the same size 1. Then, the value of the optimal solution is at most $O(mL)$. A possible $O(mL)$ schedule can be found Fig. 4.

**Lemma 8.** *Consider the instance stated in Observation 4, an $O(m)$-approximation solution can schedule at most $\frac{m}{\sqrt{L}}$ proportion of the total volume of the sequential jobs in $[0, \frac{L}{2}]$.*

**Proof.** Note that the total volume of the sequential jobs is $\frac{mL}{2}$. Assume that there are $\frac{1}{x}$ proportion of the total volume of the sequential jobs scheduled in $[0, \frac{L}{2}]$, where $x \geq 1$. Then, at least $\frac{mL}{2x}$ volume of parallel jobs is scheduled after time $\frac{L}{2}$. Since each parallel job has a size of 1, at least $\frac{mL}{2x}$ parallel jobs have completion time larger than $\frac{L}{2}$. Then, comparing the schedule stated in Observation 4, the total increase delay of these $\frac{mL}{2x}$ parallel jobs is at least $(\frac{1}{m} + \frac{L}{2x}) \cdot \frac{L}{4x}$, i.e., the latest $\frac{mL}{2x}$ parallel jobs complete at the time $\frac{L}{2} + \frac{1}{m}$. The total cost of the delay has to be bounded by $O(mL)$ for an algorithm to be $O(m)$-competitive. This yields the following claim: $(\frac{1}{m} + \frac{L}{2x}) \cdot \frac{L}{4x}$ is $O(m^2 L)$. Thus, $x \geq \frac{\sqrt{L}}{m}$. Note that $\frac{m}{\sqrt{L}}$ is small when $L \gg m$. □

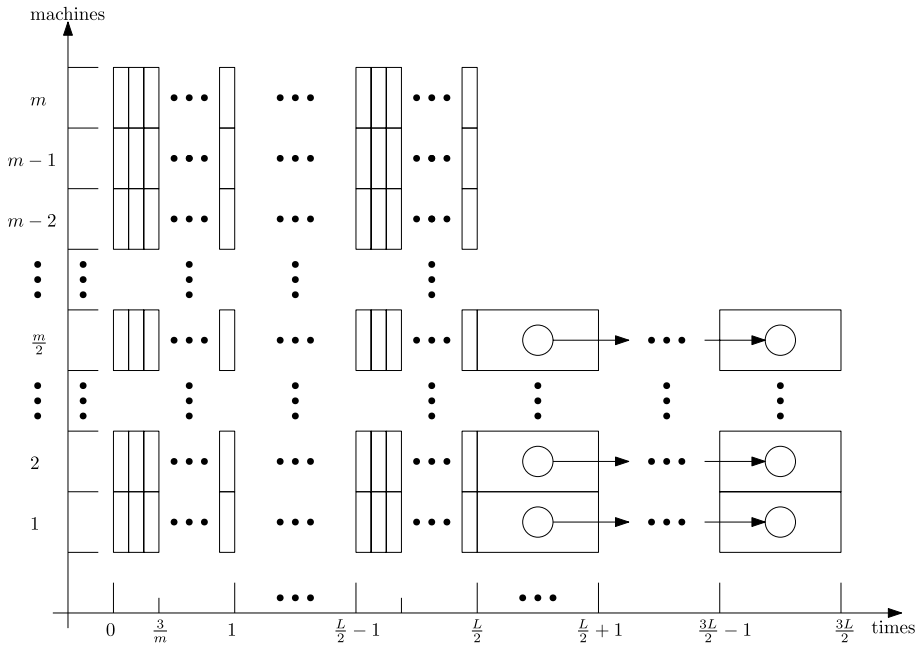


**Fig. 4.** Illustration for an $O(mL)$ solution described in Observation 4. The solution will first process the parallel jobs as long as it arrives using $m$ machines and schedule them in a way stated in Observation 3. This will take $\frac{L}{2}$ times. Then, the solution will schedule the $\frac{m}{2}$ sequential jobs using $\frac{m}{2}$ machines. Clearly, the total flow time of the jobs in $J_1$ is $\frac{3mL}{4}$ and the total flow time of the jobs in $J_2$ is $\frac{L}{2}$.

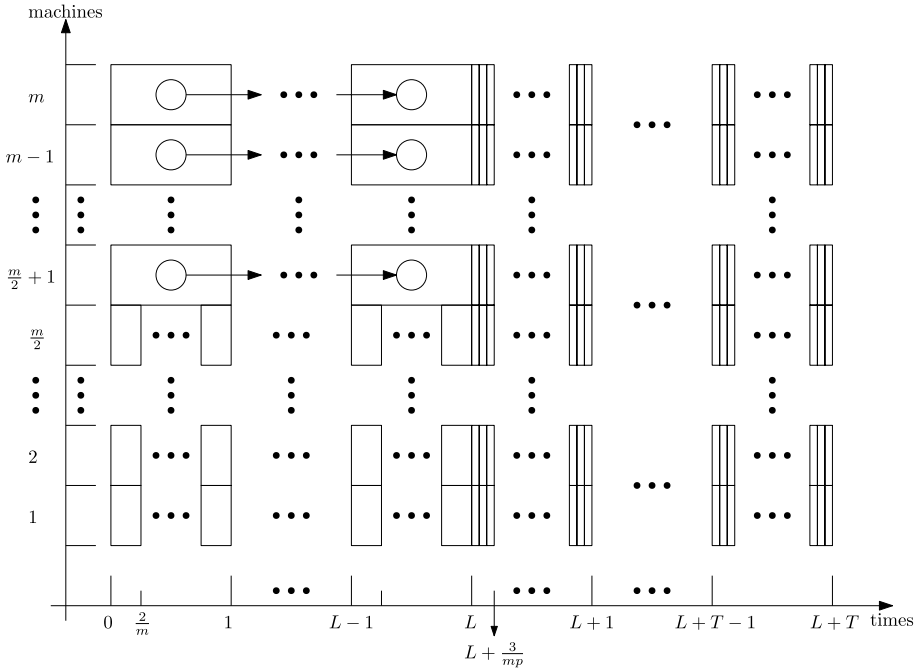


**Fig. 5.** Illustration for an $O(mL^2+T)$ schedule stated in Observation 5. The solution will first process the sequential jobs as long as they arrive using $\frac{m}{2}$ machines and schedule the parallel jobs in $J_2$ in $[0, L]$ using $\frac{m}{2}$ machines. Then, the solution will schedule the parallel jobs in $J_3$ using $m$ machines. The total flow time of the jobs in $J_1$, $J_2$, and $J_3$ is $\frac{mL}{2}$, $O(mL^2)$, and $T$, respectively.

**Observation 5.** Let $J = J_1 \cup J_2 \cup J_3$ be a job set, where $J_1$ and $J_2$ are the job sets stated in Observation 4. $J_3 = \{j_1^3, \ldots, j_i^3, \ldots\}$ contains $Tmp$ parallel jobs of size $\frac{1}{p}$, where $j_i^3$ is released at time point $L + \frac{i-1}{mp}$. Then, the value of the optimal solution is at most $O(mL^2+T)$. A possible $O(mL^2+T)$ schedule can be found in Fig. 5.

**Lemma 9.** *Considering the job set $J_3$ stated in Observation 5, an $O(m)$-competitive algorithm can schedule at most $(\frac{mL^2+T}{TpL-T^2p})$-proportion of the total volume of the jobs in $J_3$ after time $\frac{3}{2}L$.*

**Proof.** Note that the total volume of the jobs in $J_3$ is $Tm$, and the first job in $J_3$ is released at time point $L$. Assume that a $\frac{1}{x}$ proportion of the total volume of the jobs in $J_3$ are scheduled after the time $\frac{3L}{2}$. Since the size of each job in $J_3$ is $\frac{1}{p}$, at least $\frac{Tmp}{x}$ jobs in $J_3$ are completed after $\frac{3L}{2}$. Then, comparing the schedule stated in Observation 5, the total increased delay of these $\frac{Tmp}{x}$ jobs is at least $\frac{TmpL}{x} - \frac{T^2mp}{x}$. The delay is bounded by $O(m^2L^2 + mT)$. This implies that: $\frac{TmpL}{x} - \frac{T^2mp}{x} \leq m^2L^2 + Tm$. Thus, we have $\frac{1}{x} \leq \frac{mL^2+T}{TpL-T^2p}$. □

Now, we are ready to prove the first part of Theorem 1. Namely, any online algorithm has a competitive ratio of $\Omega(m)$ for average flow time in the DAG model.

**Proof of the first part of Theorem 1.** Consider the hard instance described in Observation 4 and Observation 5. By Lemma 8, we know that any $O(m)$-competitive algorithm will schedule at most $\frac{m}{\sqrt{L}}$ proportion of the total volume of the sequential jobs in $[0, \frac{L}{2}]$. Thus, in the best case, the total volume $\frac{mL}{2} - \epsilon$ of the sequential jobs will be scheduled in $[\frac{L}{2}, \frac{3L}{2}]$ when $L \gg m$. This implies that at least the volume $\frac{mL}{4} - \epsilon$ of the sequential jobs will be scheduled in $[L, \frac{3L}{2}]$. By Lemma 9, we know that at most the volume $Tm \cdot (\frac{mL^2+T}{TpL-T^2p})$ of jobs in $J_3$ can be scheduled after time $\frac{3L}{2}$. Observe that if $Tm \cdot (\frac{mL^2+T}{TpL-T^2p}) < \frac{mL}{4}$, then the lower bound $\Omega(m)$ directly follows. Clearly, setting the parameters $L > T^2$ and $p > m^2$, then the inequality above always holds. □

It is worth noting that the hard instance stated in Observation 4 and Observation 5 implies a lower bound of the speed-up curves model with the objective of the $\ell_1$-norm on flow times. Namely, any online algorithm has a competitive ratio of at least $\Omega(m)$ for average flow time in the speed-up curves model.

**Proof of the second part of Theorem 1.** Each job in the DAG instance in the proof the first part of Theorem 1 has a corresponding job in the speed-up curve instance. The parallelism of the jobs will behave identically between the two models. Thus, the lower bound extends to the speed-up curves model.

For every job in DAG model we create one job in the speed-up curves model with the same processing time. We define an appropriate concave speed-up function for the sequential jobs and parallel jobs, respectively. For every parallel job $i$, the speed-up function is defined as the inverse proportional function of the number of machines, i.e., $\Gamma_i(m') = \frac{p_i}{m'}$. For every sequential job $j$, the speed-up function is defined as the as a constant function, i.e., $\Gamma_j(m') = p_j$. Clearly, both the inverse proportional function ($m' > 0$) and the constant function are concave. Thus, the constructed instance is a valid instance for the speed-up curves model. This instance is equivalent to the instance created in the lower bound for the DAG model. □

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

Benjamin Moseley was supported in part by a Google Research Award, an Infor Research Award, a Carnegie-Bosch endowed chair, and NSF grants CCF-1824303, CCF-1845146, CCF-2121744, and CMMI-1938909. This work was done when Ruilong Zhang visited professor Benjamin Moseley at Carnegie Mellon University. The authors thank two anonymous reviewers for their comments.

## References

[1] Kunal Agrawal, I-Ting Angelina Lee, Jing Li, Kefu Lu, Benjamin Moseley, Practically efficient scheduler for minimizing average flow time of parallel jobs, in: IPDPS, IEEE, 2019, pp. 134–144.
[2] Kunal Agrawal, Jing Li, Kefu Lu, Benjamin Moseley, Scheduling parallel DAG jobs online to minimize average flow time, in: SODA, SIAM, 2016, pp. 176–189.
[3] Kunal Agrawal, Jing Li, Kefu Lu, Benjamin Moseley, Scheduling parallelizable jobs online to maximize throughput, in: LATIN, in: Lecture Notes in Computer Science, vol. 10807, Springer, 2018, pp. 755–776.
[4] Krishna P. Belkhale, Prithviraj Banerjee, An approximate algorithm for the partitionable independent task scheduling problem, in: Benjamin W. Wah (Ed.), Proceedings of the 1990 International Conference on Parallel Processing, Urbana-Champaign, IL, USA, August 1990. Vol. 1: Architecture, Pennsylvania State University Press, 1990, pp. 72–75.
[5] Jacek Blazewicz, Mikhail Y. Kovalyov, Maciej Machowiak, Denis Trystram, Jan Weglarz, Preemptable malleable task scheduling problem, IEEE Trans. Comput. 55 (4) (2006) 486–490.

[6] Maciej Drozdowski, Wieslaw Kubiak, Scheduling parallel tasks with sequential heads and tails, Ann. Oper. Res. 90 (1999) 221–246.
[7] Jianzhong Du, Joseph Y.-T. Leung, Complexity of scheduling parallel task systems, SIAM J. Discrete Math. 2 (4) (1989) 473–487.
[8] Pierre-François Dutot, Grégory Mounié, Denis Trystram, Scheduling parallel tasks approximation algorithms, in: Handbook of Scheduling, Chapman and Hall/CRC, 2004.
[9] Jeff Edmonds, Sungjin Im, Benjamin Moseley, Online scalable scheduling for the $\ell_k$-norms of flow time without conservation of work, in: SODA, SIAM, 2011, pp. 109–119.
[10] Jeff Edmonds, Kirk Pruhs, Scalably scheduling processes with arbitrary speedup curves, ACM Trans. Algorithms 8 (3) (2012) 28.
[11] Dimitris Fotakis, Jannik Matuschke, Orestis Papadigenopoulos, Malleable scheduling beyond identical machines, in: Dimitris Achlioptas, László A. Végh (Eds.), Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA, in: LIPIcs, vol. 145, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 17.
[12] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, Kirk Pruhs, Scheduling jobs with varying parallelizability to reduce variance, in: SPAA, ACM, 2010, pp. 11–20.
[13] Sungjin Im, Benjamin Moseley, Kirk Pruhs, A tutorial on amortized local competitiveness in online scheduling, SIGACT News 42 (2) (2011) 83–97.
[14] Sungjin Im, Benjamin Moseley, Kirk Pruhs, Eric Torng, Competitively scheduling tasks with intermediate parallelizability, ACM Trans. Parallel Comput. 3 (1) (2016) 4.
[15] Klaus Jansen, Scheduling malleable parallel tasks: an asymptotic fully polynomial time approximation scheme, Algorithmica 39 (1) (2004) 59–81.
[16] Klaus Jansen, Ralf Thöle, Approximation algorithms for scheduling parallel jobs, SIAM J. Comput. 39 (8) (2010) 3571–3615.
[17] Klaus Jansen, Hu Zhang, An approximation algorithm for scheduling malleable tasks under general precedence constraints, ACM Trans. Algorithms 2 (3) (2006) 416–434.
[18] Konstantin Makarychev, Debmalya Panigrahi, Precedence-constrained scheduling of malleable jobs with preemption, in: Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, Elias Koutsoupias (Eds.), Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I, in: Lecture Notes in Computer Science, vol. 8572, Springer, 2014, pp. 823–834.
[19] Gregory Mounie, Christophe Rapine, Denis Trystram, A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks, SIAM J. Comput. 37 (2) (2007) 401–412.
[20] Kirk Pruhs, Julien Robert, Nicolas Schabanel, Minimizing maximum flowtime of jobs with arbitrary parallelizability, in: WAOA, in: Lecture Notes in Computer Science, vol. 6534, Springer, 2010, pp. 237–248.
[21] John Turek, Joel L. Wolf, Philip S. Yu, Approximate algorithms scheduling parallelizable tasks, in: Lawrence Snyder (Ed.), Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '92, San Diego, CA, USA, June 29 - July 1, 1992, ACM, 1992, pp. 323–332.
[22] Deshi Ye, Danny Z. Chen, Guochuan Zhang, Online scheduling of moldable parallel tasks, J. Sched. 21 (6) (2018) 647–654.