Scheduling Parallel Jobs Online with Convex and Concave Parallelizability[*]

Roozbeh Ebrahimi[1], Samuel McCauley[**][12], and Benjamin Moseley[3]

[1] Stony Brook University, Stony Brook, NY 11794 USA
{rebrahimi,smccauley}@cs.stonybrook.edu
[2] City University of Hong Kong, Tat Chee Avenue, Kowloon, HK
[3] Washington University in St. Louis, St. Louis MO, 63130, USA
bmoseley@wustl.edu

Abstract. Online scheduling of parallelizable jobs has received a significant amount of attention recently. Scalable algorithms are known—that is, algorithms that are $(1+\varepsilon)$-speed $O(1)$-competitive for any fixed $\varepsilon > 0$. Previous research has focused on the case where each job's parallelizability can be expressed as a concave speedup curve. However, there are cases where a job's speedup curve can be convex. Considering convex speedup curves has received attention in the offline setting, but, to date, there are no positive results in the online model. In this work, we consider scheduling jobs with convex or concave speedup curves for the first time in the online setting. We give a new algorithm that is $(1+\varepsilon)$-speed $O(1)$-competitive. There are strong lower bounds on the competitive ratio if the algorithm is not given resource augmentation over the adversary, and thus this is essentially the best positive result one can show for this setting.

Keywords: Online scheduling, Convex and concave parallelizability, Competitive analysis

1 Introduction

Scheduling jobs online arises in numerous applications and, for this reason, there has been extensive research on the topic. See [14, 21] for an overview of recent work. In general, there are n jobs that arrive over time. Each job J_i has a processing time p_i and a release date r_i. Each job J_i can only be processed after its release date and is completed once it receives p_i units of processing. In the *online* setting, the scheduler is only aware of the job after it is released.

In this study, the objective is to minimize average (total) flow time, the most popular and well-studied objective in the online setting. If a scheduler A completes job J_i at time C_i^A the *flow time* for job J_i is $F_i^A = C_i^A - r_i$. This is

the amount of time the job waits to be satisfied. A scheduler A that minimizes average flow time optimizes $\sum_i F_i^A/n$, which is the average waiting time of jobs.

In the most basic setting, the jobs are to be scheduled on a single machine. In this case, it is well-known that the simple algorithm Shortest-Remaining-Processing-Time (SRPT) is optimal. Recently, there have been many results focusing on optimizing average flow time in a variety of multiple machine models; for example, see [5, 6, 11, 13]. Much of this work has focused on the case where jobs are to be scheduled on at most one machine at any point in time. However, a significant amount of attention has also been paid to scheduling jobs that are parallelizable across m identical processors.

One of the most popular models is the arbitrary speedup curves model [7], where each job J_i has a speedup function $\Gamma(x) : m \to \mathbb{R}^+$. Here $\Gamma_i(x)$ is the rate at which job J_i is processed when given x *processing units*.

It is assumed in previous work on online scheduling that the speedup Γ_i is a non-decreasing concave function. It is interesting to note that this model captures the classic identical machine scheduling setting where a job can only be processed by at most one machine any point in time. To see this, let $\Gamma_i(x) = x$ for $x \in [0, 1]$ and 1 for $x > 1$ for all jobs J_i.

Since no algorithm can be $O(1)$-competitive for the arbitrary speedup curve model [18], previous work has focused on the resource augmentation model [17], where a job is given faster processors than the adversary. An algorithm is said to be s-speed c-competitive if the algorithm is given processors of speed s, the adversary has processors of speed 1 and the competitive ratio is c. In the speedup curve setting, this can be interpreted as $\Gamma_i(x)$ being increased by a factor s for the algorithm. An ideal algorithm is one that is $(1 + \varepsilon)$-speed $O(1)$-competitive for every fixed $\varepsilon > 0$. That is, the algorithm has a constant competitive ratio while using an arbitrarily small amount of extra resources over the adversary. Such an algorithm is called *scalable*.

It was first shown that the algorithm Round Robin, or processor sharing, is $(2 + \varepsilon)$-speed $O(1)$-competitive for any fixed $\varepsilon > 0$ [7]. This was the best positive result for roughly a decade, until a breakthrough result of Edmonds and Pruhs [9] showed an algorithm, called LAPS, is scalable. This algorithm has been extremely influential since its introduction, being shown to be the best possible algorithm in numerous scheduling environments [1, 8, 12].

Recently, there has also been work on determining the best competitive ratio that can be achieved if the algorithm is not given resource augmentation [16]. In this model, for concave polynomial speedup functions, there is an $O(\log P)$-competitive algorithm that is a hybrid algorithm between SRPT and Round Robin (here P is the size of the largest job); this bound is essentially tight.

Convex Speedup Functions   We offer a model of online scheduling for arbitrary speedup curves which allows speedup functions to be convex as well as concave. All of the previous work on the online arbitrary speedup curves setting only allow speedup functions to be concave. However, several works have considered convex speedup functions in the *the malleable task model*, the offline equivalent of our setting [3, 4, 19].

Blazewicz et al. in [4] give some examples of applications of parallel computer systems in scientific computing of highly parallelizable tasks to justify the consideration of convex speedup curves. Their examples include 1) simulation of molecular dynamics, 2) Cholesky factorization, and 3) operational oceanography.

For completeness, we summarize the molecular dynamics problem and why it experiences convex speedup. Simulation of molecular dynamics of large organic molecules (like proteins) is usually a very complicated task, because the simulation must calculate interactions between hundreds of thousands of atoms at each step. These simulations require massive amounts of space, and when an instance does not fit in main memory the cost of repeatedly writing/reading from disk dictates the time of execution. As a result, increasing the number of processors working on the problem decreases these disk accesses. This leads to a convex speedup (see Figure 1 in [4]).

Several other studies report convex speedup curves in practice. For example, Beaumont and Guermouche in [2] report that implementing the sparse matrix factorization method of Prasanna and Musicus [20] in a real multifrontal solver results in a measured speedup function of $p^{1.15}$. They state that this superlinear speedup originates from unusual requirements on processor utilization: the algorithm generates master and slave tasks which must be scheduled on different processors.

Our Contributions   In this work, we consider scheduling jobs in the online setting with speedup curves that may be convex or concave for the first time. In our model, each job J_i is comprised of phases $\langle J_i^1, J_i^2, \cdots, J_i^{q_i} \rangle$ and each phase can have an arbitrary speedup curve. We assume that the parallelizability of phase μ of job i, J_i^μ is either a convex or concave function and each job must be assigned an integral number of processing units at each time. Note that each phase of each job can have different parallelizability, and can change from convex to concave or visa versa. Since this setting is more general than the parallel identical machines setting, there are strong lower bounds on the competitive ratio of any online algorithm.[4] Thus, we consider algorithms in the resource-augmentation model.

We assume the scheduler is online and does not know of a job until it arrives. We further assume that the scheduling algorithm does not know the size of a job nor the specific function of its speedup curve. However, we assume that the the algorithm knows whether $\Gamma_{i,\mu}$ for each phase μ of job i, J_i^μ is convex or concave. That is, while that algorithm unaware of $\Gamma_{i,\mu}$ for each job J_i and phase μ, it is aware if $\Gamma_{i,\mu}$ is convex or concave. We refer to such an scheduler as a **convexity-sensitive scheduler**. We note that a convexity-sensitive scheduler has only *one bit per phase* of each job more information than non-clairvoyant LAPS algorithm in [9]. Here we show the following result.

Theorem 1. *There exists a $(1+\lambda\varepsilon)$-speed $O(\frac{1}{\lambda\varepsilon^4})$-competitive* convexity-sensitive *scheduling algorithm for minimizing average flow time in the arbitrary speedup*

---

[4] Specifically, [18] gives an $\Omega(\log n/m)$ lower bound.

curves setting when the speedup curves can be a concave or convex function, for any $0 < \varepsilon \leq 1/2$ and $\lambda > 7/3$.

Thus, for any fixed ε', we can achieve a $(1 + \varepsilon')$ speed, $O(1)$-competitive algorithm by appropriately choosing ε and λ in the above theorem. Given the strong lower bound on the competitive ratio for any algorithm without resource augmentation, this is essentially the best positive result that can be shown in the worst-case analysis setting.

Organization   Section 2 outlines some preliminary tools for our analysis. In Section 3, we present our convexity-sensitive algorithm. Section 4 provides the proof for Theorem 1.

2   Preliminaries

In our setup we consider n jobs that arrive over time. Each job J_i has a **release time** $\boldsymbol{r_i}$ and a sequence of phases $\chi_i = \langle J_i^1, J_i^2, \cdots, J_i^{q_i} \rangle$. Each phase μ of job i has a **processing time** $\boldsymbol{p_{i,\mu}}$ which means that it needs this amount of processing to be completed. We assume that there are m available processing units and that when phase μ of job i is given x (*integral*) units of processing it is processed at a rate of $\boldsymbol{\Gamma_{i,\mu}(x)}$. We assume that $\Gamma_{i,\mu}(0) = 0$. If the algorithm is given **resource augmentation** $\boldsymbol{s}$, then we assume $\Gamma_{i,\mu}(x)$ is scaled by s for the algorithm (whose performance is then compared to the unscaled optimal solution).

Our algorithm assigns fractional units of processing units to jobs, which might seem to contradict the model described above. We describe how this assumption is justifiable. First, since $\Gamma_{i,\mu}(x)$ is only defined on integral x, we can extend the definition to where x can be fractional. This is naturally defined by making $\Gamma_{i,\mu}$ a piecewise linear function. Consider any fractional $x = x' + \lambda$ where $\lambda \in (0, 1)$ and x' is fractional. We assume that $\Gamma_{i,\mu}(x) = (1 - \lambda)\Gamma_{i,\mu}(x') + \lambda\Gamma_{i,\mu}(x' + 1)$. Note that this preserves convexity and concavity of $\Gamma_{i,\mu}$. In this case, [10] has shown that an online $O(c)$-competitive algorithm that fractionally assigns processors to jobs can be converted to an $O(c)$-competitive online algorithm that integrally assigns processors to the jobs without knowing the functions $\Gamma_{i,\mu}$. In short, one can simulate fractional processor assignments by assigning jobs to an additional processor for a small amount of time, leading to equivalent speedup. Thus, we assume WLOG that our algorithm can assign processors fractionally to the jobs. Furthermore, by scaling $\Gamma_{i,\mu}$, we may assume that $\boldsymbol{m = 1}$ throughout the analysis.

We will also extend this problem definition to allow phases of jobs J_i^μ where $\boldsymbol{\Gamma_{i,\mu}(x) = 1}$ **for all** $\boldsymbol{x \in [0, \infty)}$. We note that this implies that phase μ of job i, J_i^μ is processed even though no processor is assigned to it. Although this is not realistic in practice, this only makes our problem harder, yet, in fact, it will help to simplify our analysis. We will call such phases **sequential**. Note that in this case, these phases are not explicitly identified to the algorithm (since it is convexity-sensitive and does not know the functions $\Gamma_{i,\mu}$), so the algorithm may waste processing power on them.

The following simple proposition about concave and convex functions will be useful throughout the analysis.

Proposition 1. – *For any positive value x, any positive $\alpha < 1$, and any concave function f where $f(0) = 0$, we have that $\alpha f(x) \leq f(\alpha x)$.*
- *Also, for any values a and b where $b \geq a > 0$, we have that $\frac{f(b)}{f(a)} \leq \frac{b}{a}$.*
- *Likewise, for any positive value x, any positive value $\alpha < 1$, and any convex function g where $g(0) = 0$, we have that $\alpha g(x) \geq g(\alpha x)$.*
- *Also, we have that for $b \geq a > 0$ $\frac{g(b)}{g(a)} \geq \frac{b}{a}$.*

2.1 Amortized Local Competitiveness

We let $F^A(I)$ and $F^O(I)$ refer to the objective values of the algorithm and OPT on input I. A scheduling algorithm is said to be **d-competitive** if

$$\max_I \frac{F^A(I)}{F^O(I)} \leq d.$$

To prove the competitiveness of an online algorithm, we use an *amortized local competitiveness argument.* See [15] for a tutorial on this technique. To incorporate such an argument it suffices to show for an algorithm A that a potential function $\Phi(t)$ with the following properties exists, where $\Phi(t)$ is continuous at all times except possibly when jobs arrive or are completed:

Boundary condition: Before any job is released $\Phi(0) = 0$, and after all jobs are finished $\Phi(\infty) \geq 0$.

Completion condition: Summing over all job completions by the optimal solution and the algorithm, Φ does not increase by more than $\alpha_1 F^O$ for some $\alpha_1 \geq 0$.

Arrival condition: Summing over the arrival of all jobs, Φ does not increase by more than $\alpha_2 F^O$ for some $\alpha_2 \geq 0$.

Running condition: At any time t when no job arrives nor is completed,

$$\frac{\partial F^A(t)}{\partial t} + \frac{\partial \Phi(t)}{\partial t} \leq \alpha_3 \frac{\partial F^O(t)}{\partial t}. \tag{1}$$

By integrating Equation 1 over time and applying the boundary, arrival, and completion conditions we get

$$F^A - \Phi(0) + \Phi(\infty) \leq (\alpha_1 + \alpha_2 + \alpha_3)F^O \tag{2}$$
$$F^A \leq (\alpha_1 + \alpha_2 + \alpha_3)F^O - \Phi(\infty).$$

Equation 2 implies that algorithm A is $(\alpha_1 + \alpha_2 + \alpha_3)$-competitive.

3 A Convexity-Sensitive Scheduling Algorithm

In this section, we present a *convexity-sensitive* scheduling algorithm that is $O(1/\lambda\varepsilon^4)$-competitive with $(1 + \lambda\varepsilon)$-speed augmentation for every $0 < \varepsilon \leq 1/2$ and $\lambda > 7/3$. Note that ε can be set independent of λ. For the remaining portion of the algorithm definition and analysis, fix a pair of such λ and ε.

We introduce some definitions and then present our scheduling algorithm.

Definition 1. *We say that phase μ of job i, J_i^μ is **sequential** if $\forall x \geq 0$, $\Gamma_{i,\mu}(x) = 1$. A **linear** phase of job j is a phase J_j^μ with speed-up function $\Gamma_{j,\mu}(x) = x$. We define both of these phases to be concave.*

Definition 2. *Fix a unit time step at time t. For a given scheduling algorithm, we let $\boldsymbol{A(t)}$ be the set of unsatisfied jobs at time t. Also let $\boldsymbol{A'(t)}$ be the $\varepsilon A(t)$ latest arriving jobs. Let $(1 - \gamma_t)$ be the fraction of alive jobs that are currently in a convex phase at time t and a γ_t fraction are in a concave phase at time t.*

Algorithm 1 The Convexity-sensitive Scheduling Algorithm.
***
1: On each time step t do the following:
2: Give each of the $(1 - \gamma_t)|A'(t)|$ jobs in a convex phase *all of the processing units* for $1/|A'(t)|$ fraction of the unit time slot.
3: Give each of the jobs in a concave phase $1/(\gamma_t |A'(t)|)$ of the processing units for a γ_t fraction of the unit time slot.

---

4 Proof of Theorem 1

We first define our potential function in subsection 4.1. Then, in subsection 4.2 we argue the *amortized local competitiveness* conditions for our potential function, proving that the convexity-sensitive scheduling algorithm is $O(1)$-competitive.

4.1 The Potential Function Φ

Definition 3. *Let $\boldsymbol{p_{i,\mu}^A(t)}$, and $\boldsymbol{p_{i,\mu}^O(t)}$ be the remaining processing time for J_i's μ-th phase in the algorithm's schedule, and OPT's schedule at time t respectively.*

If job J_i is processed using x_i processing units by the algorithm when it is in phase μ in the algorithm's schedule at time t then $p_{i,\mu}^A(t)$ decreases at a rate of $(1 + \lambda \varepsilon)\Gamma_{i,\mu}(x_i)$. Similarly, if job J_i is processed using x_i processing units by the optimal solution in phase μ at time t then $p_{i,\mu}^O(t)$ decreases at a rate of $\Gamma_{i,\mu}(x_i)$

Definition 4. *We define the **lag** of the algorithm on job J_i's μ-th phase, J_i^μ, compared to OPT as following:*

$$z_{i,\mu}(t) = \max\{p_{i,\mu}^A(t) - p_{i,\mu}^O(t), 0\}.$$

Note that $z_{i,\mu}$ is never negative.

Definition 5. *We define the **rank of job J_i**, $\boldsymbol{rank_i(t)}$, as the number of jobs in $A(t)$ that arrived before job J_i in the system.*

Definition 6. *We define $\boldsymbol{\beta_{i,\mu}(x)}$ to be equal to*

$$\beta_{i,\mu}(x) = \begin{cases} \Gamma_{i,\mu}\left(1/\varepsilon x\right) & \text{if } \Gamma_{i,\mu} \text{ is concave;} \\ \Gamma_{i,\mu}(1)/\varepsilon x & \text{if } \Gamma_{i,\mu} \text{ is convex.} \end{cases}$$

Finally, we are ready to define our potential function.

Definition 7. *Let $c = 20/\varepsilon^2(9\lambda - 21)$. We define the potential function $\boldsymbol{\Phi(t)}$*

$$\Phi(t) = c \sum_{i \in A(t)} \sum_{\mu \in \chi_i} \frac{z_{i,\mu}(t)}{\beta_{i,\mu}(\boldsymbol{rank_i}(t))}.$$

4.2 Amortized Local Competitiveness of $\boldsymbol{\Phi}$

In this subsection, we show that our potential function Φ satisfies the four conditions laid out in the Section 2 and thus the convexity-sensitive scheduling algorithm is constant competitive.

The following lemma shows that the algorithm has the first three conditions. Its proof is rather straightforward and is omitted due to space constraints.

Lemma 1. *The potential function, $\Phi(t)$, satisfies the* boundary, arrival *and* completion *conditions. In particular, the potential does not increase at any of these events.*

To prove Theorem 1, we need to show the **running condition** for Φ. To do this, we show that we can focus on certain classes of problem instances. The following is a simple extension of a lemma proven in [9].

Lemma 2. *Let S be a* convexity-sensitive scheduler *with s-speed augmentation. Let I be an instance of jobs with phases that have concave or convex speed up functions. There exists an instance I' that includes the same set of convex phases for jobs in I, and for every concave phase J_i^μ in I' it is the case that either J_i^μ is* sequential *($\Gamma_{i,\mu} = 1$), or linear ($\Gamma_{i,\mu}(x) = x$). Furthermore, such an I' exists where $F^S(I) = F^S(I')$ and $F^O(I') \leq F^O(I)$ where F^O is the objective of the optimal solution and F^S is the objective of the convexity-sensitive scheduler.*

The proof of the above lemma is implied immediately by the proof of Lemma 3.1 in [9]. Specifically, the proof proceeds constructively, by substituting each *concave* phase that is not sequential or linear with one that is; in each case, the objective of the optimal solution decreases, while that of the convexity-sensitive scheduler stays the same. Our construction does not change the convex phases.

This lemma implies that if an instance has concave phases that are not sequential or linear, the performance of our algorithm can only improve relative to the optimal solution. Thus we assume throughout the proof that concave phases are either sequential or linear.

Definition 8. *Let $\boldsymbol{U(t)}$ be the set of unsatisfied jobs in the optimal solution, OPT, at time t.*

First we show how much Φ can increase at time t due to the optimal solution processing jobs. Then, later, we bound the decrease due to the algorithm processing separately.

Recall that for total flow time, the increase in the objective at any point in time is the number of unsatisfied jobs. Thus, for an instantaneous time t we have

$$\frac{\partial F^A(t)}{\partial t} = |A(t)|, \ \frac{\partial F^O(t)}{\partial t} = |U(t)|.$$

Lemma 3. *The total increase in $\Phi(t)$ at any time t due to the optimal solution processing jobs is at most $c|U(t)| + c\varepsilon|A(t)|$.*

Proof. Our goal is to show that the optimal solution cannot increase the potential function too much. To show this, consider the number of processing units the optimal solution assigns to the jobs.

Definition 9. *Let $m_i^O(t)$ be the number of processing units OPT assigns to job J_i at time t.*

Let $\mu_i^O(t)$ be the phase of job J_i in OPT's schedule at time t. Job J_i is processed by OPT at the rate of $\Gamma_{i,\mu_i^O(t)}(m_i^O(t))$ (the remaining processing time $p_{i,\mu_i^O(t)}^O(t)$ of phase $\mu_i^O(t)$ for job i would *decrease* at this rate). Then $z_{i,\mu_i^O(t)}(t)$ could *increase* by this amount in the worst case. Hence,

$$\frac{\partial \Phi(t)}{\partial t} \le c \sum_{i \in U(t)} \frac{\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\beta_{i,\mu_i^O(t)}(\mathtt{rank}_i(t))}.$$

Let $U_v(t)$ be the alive jobs in OPT at time t that are in a *convex* phase at time t in OPT's schedule and $U_c(t)$ be ones that are in a *concave* phase.

$$\frac{\partial \Phi(t)}{\partial t} \le c \sum_{i \in U(t)} \frac{\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\beta_{i,\mu_i^O(t)}(\mathtt{rank}_i(t))}$$

$$\le c \left(\sum_{i \in U_c(t)} \frac{\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\beta_{i,\mu_i^O(t)}(\mathtt{rank}_i(t))} + \sum_{i \in U_v(t)} \frac{\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\beta_{i,\mu_i^O(t)}(\mathtt{rank}_i(t))} \right)$$

$$\le c \left(\sum_{i \in U_c(t)} \frac{\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\Gamma_{i,\mu_i^O(t)}(1/\varepsilon\mathtt{rank}_i(t))} + \varepsilon \sum_{i \in U_v(t)} \frac{\mathtt{rank}_i(t)\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\Gamma_{i,\mu_i^O(t)}(1)} \right).$$

We now bound each of these terms separately.

First summation term If $m_i^O(t) \le 1/(\varepsilon\mathtt{rank}_i(t))$, then the corresponding term in the first summation is at most 1. Note that this is true even if the job is in a sequential phase and we can assume OPT does not assign any processors to sequential jobs since it does not increase the rate they are processed.

On the other hand, if $m_i^O(t) > 1/(\varepsilon\mathtt{rank}_i(t))$, then by Proposition 1, we have

$$\frac{\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\Gamma_{i,\mu_i^O(t)}(1/\varepsilon\mathtt{rank}_i(t))} \le \varepsilon\mathtt{rank}_i(t)m_i^O(t).$$

due to the concavity of any job J_i's phase in the first sum. Therefore,

$$\sum_{i \in U_c(t)} \frac{\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\Gamma_{i,\mu_i^O(t)}(1/\varepsilon \texttt{rank}_i(t))} \leq |U_c(t)| + \sum_{i \in U_c(t)} \varepsilon \texttt{rank}_i(t) m_i^O(t).$$

Second summation term As for the second term we know that $m_i^O(t) \leq 1$. Hence, due to the convexity of the jobs phases, Proposition 1 implies that

$$\frac{\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\Gamma_{i,\mu_i^O(t)}(1)} \leq m_i^O(t).$$

Thus, we get that

$$\sum_{i \in U_v(t)} \frac{\texttt{rank}_i(t)\Gamma_{i,\mu_i^O(t)}(m_i^O(t))}{\Gamma_{i,\mu_i^O(t)}(1)} \leq \sum_{i \in U_v(t)} \texttt{rank}_i(t) m_i^O(t).$$

Substituting the above simplifications, we get

$$\frac{\partial \Phi(t)}{\partial t} \leq c\varepsilon \texttt{rank}_i(t) \left(\sum_{i \in U_c(t)} m_i^O(t) + \sum_{i \in U_v(t)} m_i^O(t) \right) + c|U_c(t)|$$

$$\leq c|U_c(t)| + c\varepsilon|A(t)| \sum_{i \in U(t)} m_i^O(t).$$

The last line follows because the rank of each job is at most $A(t)$ at time t, by definition of the ranks of the jobs and because there are only $A(t)$ unsatisfied jobs in the algorithm's schedule at time t.

Finally, we know that $\sum_{i \in U(t)} m_i^O(t) \leq 1$, because the amount of processing units divided between jobs at any time can not exceed $m = 1$. Hence,

$$\frac{\partial \Phi(t)}{\partial t} \leq c|U_c(t)| + c\varepsilon|A(t)| \leq c|U(t)| + c\varepsilon|A(t)|.$$

$$\square$$

We divide the rest of the analysis into two cases depending on the relationship between $U(t)$ and $A(t)$. First, we consider the easier case where $|U(t)| \geq \varepsilon^2 |A(t)|/10$.

Lemma 4. *If $|U(t)| \geq \varepsilon^2 |A(t)|/10$, then $\partial F^A(t)/\partial t + \partial \Phi(t)/\partial t$ is at most $O(1/\lambda\varepsilon^4) \left(\partial F^O(t)/\partial t\right)$.*

Proof. By Lemma 3 we know that the most OPT can increase $\Phi(t)$ is $c|U(t)| + c\varepsilon|A(t)|$. Hence,

$$c|U(t)| + c\varepsilon|A(t)| \leq c(1 + 10/\varepsilon)|U(t)| \qquad \text{since } \varepsilon|A(t)| \leq 10|U(t)|/\varepsilon.$$

9

Therefore we have,

$$\frac{\partial F^A(t)}{\partial t} + \frac{\partial \Phi(t)}{\partial t} \leq A(t) + c(1 + 10/\varepsilon)|U(t)|$$

$$\leq c(1 + 10/\varepsilon + 10/\varepsilon^2)|U(t)| \qquad \text{since } |A(t)| \leq 10|U(t)|/\varepsilon^2,$$

$$\leq O(1/\lambda\varepsilon^4)\left(\frac{\partial F^O(t)}{\partial t}\right) \qquad \text{since } c = \frac{20}{\varepsilon^2(9\lambda - 21)}.$$

Now we consider the more challenging case where $|U(t)| < \varepsilon^2|A(t)|/10$.

Lemma 5. *If $|U(t)| < \varepsilon^2|A(t)|/10$, then $\partial F^A(t)/\partial t + \partial \Phi(t)/\partial t$ is at most $O(1/\lambda\varepsilon^4)\left(\partial F^O(t)/\partial t\right)$.*

Proof. In the proof of Lemma 4 we focused on how OPT can increase Φ (by assuming that the algorithm didn't decrease the $p^A_{i,\mu^O_i(t)}(t)$ variables at all). Let $\mu^A_i(t)$ be the phase job J_i is in at time t in the algorithm's schedule. In this proof, we focus on how the algorithm can decrease $z_{i,\mu^A_i(t)}(t)$ and thus $\Phi(t)$ as well as OPT increasing $\Phi(t)$. Let $\boldsymbol{\mathcal{C}^O(t)}$ be the change in $\Phi(t)$ due to the optimal solution processing jobs at time t and let $\boldsymbol{\mathcal{C}^A(t)}$ denote the change in $\Phi(t)$ due to the algorithm processing of jobs at time t. Lemma 3 says that $\mathcal{C}^O(t) \leq c|U(t)| + c\varepsilon|A(t)|$.

Now, we bound $\mathcal{C}^A(t)$ at a time t where $|U(t)| < \varepsilon^2|A(t)|/10$. Recall that $z_{i,\mu^A_i(t)}(t) = \max\{p^A_{i,\mu^A_i(t)}(t) - p^O_{i,\mu^A_i(t)}(t), 0\}$. Therefore, $z_{i,\mu^A_i(t)}$ can only decrease due to the algorithm's processing. Further, $z_{i,\mu^A_i(t)}(t)$ will decrease at the rate the algorithm process job J_i at time t if the optimal solution has completed J_i by time t. That is, for jobs not in $U(t)$. Since OPT only has $U(t) < \varepsilon^2|A(t)|/10$ unfinished jobs, the algorithm's processing on at least a $(1 - \varepsilon/10)$ fraction of the jobs in $A'(t)$ causes $z_{i,\mu^A_i(t)}$ to decrease at the rate they are processed.

Let $A_c(t)$ be the set of jobs in $A'(t)$ that are in a concave phase at time t in the algorithm's schedule. Let $A_v(t)$ be the set of jobs in $A'(t)$ that are in a convex phase at time t in the algorithm's schedule. $A'(t) = A_c(t) \cup A_v(t)$. We assume that at time t a $(1 - \gamma_t)$ fraction of jobs in $A'(t)$ are in $A_v(t)$ and a γ_t fraction are in $A_c(t)$

By Lemma 2, we may assume that any concave phase for a job is either *sequential* or *linear*. Therefore, to bound $\mathcal{C}^A(t)$ we can assume that all the jobs in $A_c(t)$ are either sequential or linear. Let $S_c(t)$ be the jobs in $A_c(t)$ that are in a sequential phase at time t and the others are in $L_c(t)$. $A_c(t) = S_c(t) \cup L_c(t)$.

Here are the progress rates of the algorithm on $A_v(t)$, $S_c(t)$, and $L_c(t)$.

- The algorithm processes each convex job in $A_v(t)$ at a rate of $\Gamma_{i,\mu^A_i(t)}(1)/\varepsilon|A(t)|$.
- For each linear job in $L_c(t)$, the rate of progress is $\gamma_t\Gamma_{i,\mu^A_i(t)}(1/\gamma_t\varepsilon|A(t)|) = 1/\varepsilon|A(t)|$.
- For each sequential job in $S_c(t)$, the rate of progress is always 1 no matter how many processing units are assigned to the job (even if there are 0 units assigned).

10

The algorithm is $(1 + \lambda\varepsilon)$-speed augmented; therefore we multiply its change in the potential function by $(1 + \lambda\varepsilon)$. Combining all the above, we get

$$
\mathcal{C}(t)^A \leq -c(1 + \lambda\varepsilon) \left(\sum_{i \in A_v(t) \setminus U(t)} \frac{\Gamma_{i,\mu_i^A(t)}(1)/\varepsilon|A(t)|}{\beta_{i,\mu_i^A(t)}(\text{rank}_i(t))} + \sum_{i \in S_c(t) \setminus U(t)} \frac{1}{\beta_{i,\mu_i^A(t)}(\text{rank}_i(t))} \right.
$$
$$
\left. + \sum_{i \in L_c(t) \setminus U(t)} \frac{1/\varepsilon|A(t)|}{\beta_{i,\mu_i^A(t)}(\text{rank}_i(t))} \right). \tag{3}
$$

Note that since the algorithm works on the latest $A'(t)$ arriving jobs, the rank of each job in $A'(t)$ is bounded between

$$
(|A(t)| - |A'(t)|) \leq \text{rank}_i(t) \leq |A(t)|
$$
$$
(1 - \varepsilon)|A(t)| \leq \text{rank}_i(t) \leq |A(t)|.
$$

By starting from Inequality 3, replacing the definition of $\beta_{i,\mu_i^A(t)}$, and using the above bounds on the rank of each job in $A'(t)$, we can show the following proposition. The proof is omitted.

Proposition 2. *Let $\mathcal{C}^A(t)$ be the change in $\Phi(t)$ due to the algorithm processing of jobs at time t. For $\varepsilon < 1/2$ and $|U(t)| < \varepsilon^2|A(t)|/10$ we have*

$$
\mathcal{C}^A(t) \leq -c\varepsilon|A(t)| \left(1 + (9\lambda - 21)\frac{\varepsilon}{20} \right).
$$

Using Proposition 2, we can upper bound $\partial\Phi(t)/\partial t$.

$$
\frac{\partial\Phi(t)}{\partial t} = \mathcal{C}^O(t) + \mathcal{C}^A(t)
$$
$$
\leq (c|U(t)| + c\varepsilon|A(t)|) - \left(c\varepsilon|A(t)| + c\varepsilon^2 \left(\frac{9\lambda - 21}{20} \right) |A(t)| \right)
$$
$$
\leq c|U(t)| - c\varepsilon^2 \left(\frac{9\lambda - 21}{20} \right) |A(t)|.
$$

With this we get the **running** condition (Equation 1) which finishes the proof of Lemma 5:

$$
\frac{\partial F^A(t)}{\partial t} + \frac{\partial\Phi(t)}{\partial t} \leq c|U(t)| + |A(t)| \left(1 - c\varepsilon^2 \left(\frac{9\lambda - 21}{20} \right) \right)
$$
$$
\leq c|U(t)| = c \left(\frac{\partial F^O(t)}{\partial t} \right) \qquad \text{since } c = \frac{20}{\varepsilon^2(9\lambda - 21)}, \ \lambda > 7/3.
$$

$$\square$$

Lemmas 4 and 5 together imply the running condition and complete the proof of Theorem 1.

5 Acknowledgements

References

1. Bansal, N., Krishnaswamy, R., Nagarajan, V.: Better scalable algorithms for broadcast scheduling. In: Proceedings of the Thirty-Seventh Annual International Colloquium on Automata, Languages, and Processing (ICALP). pp. 324–335 (2010)
2. Beaumont, O., Guermouche, A.: Task scheduling for parallel multifrontal methods. In: Euro-Par Parallel Processing, pp. 758–766. Springer (2007)
3. Blazewicz, J., Kovalyov, M.Y., Machowiak, M., Trystram, D., Weglarz, J.: Preemptable malleable task scheduling problem. IEEE Transactions on Computers 55(4), 486–490 (2006)
4. Blazewicz, J., Machowiak, M., Weglarz, J., Kovalyov, M.Y., Trystram, D.: Scheduling malleable tasks on parallel processors to minimize the makespan. Annals of Operations Research 129(1-4), 65–80 (2004)
5. Chadha, J.S., Garg, N., Kumar, A., Muralidhara, V.N.: A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In: Proceedings of the 41st Symposium on Theory of Computation (STOC) (2009)
6. Chan, S.H., Lam, T.W., Lee, L.K., Zhu, J.: Nonclairvoyant sleep management and flow-time scheduling on multiple processors. In: Proceedings of the 25th Symposium on Parallelism in Algorithms and Architectures (SPAA). pp. 261–270 (2013)
7. Edmonds, J.: Scheduling in the dark. Theoretical Computer Science 235(1), 109–141 (2000 Preliminary version in $STOC$ 1999)
8. Edmonds, J., Im, S., Moseley, B.: Online scalable scheduling for the ℓ_k-norms of flow time without conservation of work. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (2011)
9. Edmonds, J., Pruhs, K.: Scalably scheduling processes with arbitrary speedup curves. ACM Transactions on Algorithms 8(3), 28:1–28:10 (Jul 2012)
10. Fox, K., Im, S., Moseley, B.: Energy efficient scheduling of parallelizable jobs. In: Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 948–957 (2013)
11. Fox, K., Moseley, B.: Online scheduling on identical machines using SRPT. In: Proceedings of the 22nd ACM Symposium on Discrete Algorithms (SODA) (2011)
12. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling jobs with varying parallelizability to reduce variance. In: Proceedings of the Twenty-Second Syposium on Parallel Algorithms and Architectures (SPAA). pp. 11–20 (2010)
13. Im, S., Moseley, B.: Online scalable algorithm for minimizing ℓ_k-norms of weighted flow time on unrelated machines. In: Proceedings of the Twenty-Second Annual ACM Symposium on Discrete Algorithms (SODA). pp. 95–108 (2011)
14. Im, S., Moseley, B., Pruhs, K.: A tutorial on amortized local competitiveness in online scheduling. SIGACT News 42, 83–97 (June 2011)
15. Im, S., Moseley, B., Pruhs, K.: A tutorial on amortized local competitiveness in online scheduling. SIGACT News 42(2), 83–97 (Jun 2011)
16. Im, S., Moseley, B., Pruhs, K., Torng, E.: Competitively scheduling tasks with intermediate parallelizability. In: Proceedings of the Twenty-Sixth ACM Symposium on Parallelism in Algorithms and Architectures (SPAA). pp. 22–29 (2014)

17. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. Journal of the ACM 47(4), 617–643 (2000)
18. Leonardi, S., Raz, D.: Approximating total flow time on parallel machines. Journal of Computer and Systems Sciences 73(6), 875–891 (2007)
19. Ludwig, W., Tiwari, P.: Scheduling malleable and nonmalleable parallel tasks. In: Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 167–176 (1994)
20. Prasanna, G.N.S., Musicus, B.R.: Generalized multiprocessor scheduling and applications to matrix computations. IEEE Trans. Parallel Distrib. Syst. 7(6), 650–664 (Jun 1996)
21. Pruhs, K., Sgall, J., Torng, E.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis, chap. Online Scheduling. CRC press (2004)