Precedence-constrained Scheduling of Malleable Jobs with Preemption[⋆]

Konstantin Makarychev[1] and Debmalya Panigrahi[⋆⋆][2]

[1] Microsoft Research, Redmond, WA; komakary@microsoft.com
[2] Duke University, Durham, NC; debmalya@cs.duke.edu

Abstract. Scheduling jobs with precedence constraints on a set of identical machines to minimize the total processing time (makespan) is a fundamental problem in combinatorial optimization. In practical settings such as cloud computing, jobs are often *malleable*, i.e., can be processed on multiple machines simultaneously. The instantaneous processing rate of a job is a non-decreasing function of the number of machines assigned to it (we call it the processing function). Previous research has focused on practically relevant concave processing functions, which obey the law of diminishing utility and generalize the classical (non-malleable) problem. Our main result is a $(2 + \epsilon)$-approximation algorithm for concave processing functions (for any $\epsilon > 0$), which is the best possible under complexity theoretic assumptions. The approximation ratio improves to $(1 + \epsilon)$ for the interesting and practically relevant special case of power functions, i.e., $p_j(z) = c_j \cdot z^\gamma$.

1 Introduction

In the *precedence-constrained scheduling* problem (we call it the PS problem), the goal is to schedule a set of jobs with precedence constraints on a set of identical machines so as to minimize the overall time for processing them (called the *makespan*). One of the first results in approximation algorithms was a 2-approximation for this problem due to Graham in 1966 [13]. On the negative side, this problem was shown to be NP-hard to approximate to a ratio better than 4/3 by Lenstra and Rinnooy Kan in 1978 [19]. In spite of substantial effort, the gap between these two bounds remained open for three decades. Recently, Svensson [27] has provided strong evidence that improving Graham's result might in fact be impossible by showing that it is tight under certain complexity theoretic assumptions.

A natural generalization of the PS problem considered in the literature is that of *malleable* jobs, i.e., jobs that can be processed simultaneously on multiple machines. This is particularly relevant in practice for domains such as cloud computing, operating systems, high performance computing, project management, etc. where a fixed set of resources must be distributed among precedence-constrained tasks to complete them by

the earliest possible time. At any given time, the processing rate of a job is a function of the number of machines assigned to it (we call it the *processing function*). The goal is to produce a schedule of minimum makespan.

Formally, the input comprises a *directed acyclic graph* (DAG) $G = (J, E)$, where each vertex $j \in J$ represents a job j and has a given size $s_j > 0$. The arcs in E represent the *precedence constraints* on the jobs, i.e., if $(j_1, j_2) \in E$, then job j_1 has to be completed before job j_2 can be processed. Let m denote the number of identical machines on which these jobs have to be scheduled. We are also given processing functions for the jobs $p_j : \{0, 1, 2, \ldots, m\} \to \mathbb{R}_0^+$ that map the number of machines assigned to the rate at which the job is processed. (Clearly, $p_j(0) = 0$ for all processing functions.)

The output of the algorithm is a schedule A, which is represented by a continuum of functions $A_t(j)$ over time $t > 0$. $A_t(j)$ represents the number of machines allocated to job j at time t. The schedule must satisfy:

- *Capacity constraints*: For any time $t \in (0, \infty)$, the number of allocated machines at time t is at most m, i.e., $\sum_{j \in J} A_t(j) \leq m$.
- *Precedence constraints*: For any arc $(j_1, j_2) \in E$ and any time $t \in (0, \infty)$, if $A_t(j_2) > 0$, then the job j_1 must be finished by time t, i.e. $\int_0^t p_{j_1}(A_{t'}(j_1))dt' \geq s_{j_1}$. Let the set of jobs that can be processed at a given time (i.e., all their predecessors in G have been completely processed) be called the set of *available* jobs. Then, the precedence constraints enforce that the schedule picks a subset of available jobs to process at any given time.

The *makespan* (or length) of the schedule is defined as the time when all jobs finish processing, i.e., $\ell(A) = \sup\{t : \sum_{j \in J} A_t(j) > 0\}$. The objective of the algorithm is to minimize the makespan of the schedule. We call this the *generalized precedence-constrained scheduling* or GPS problem.

Preemption. It is important to note that we allow *preemption*, i.e., at any point of time, the remaining volume of an available job can be scheduled on any number of machines independent of the history of where it was processed earlier. Therefore, our schedule is defined simply by the number of machines allocated to a job at any given time, and not by the identities of the machines themselves. This is a departure from the bulk of the existing literature in precedence-constrained scheduling with malleable jobs, where preemption is typically disallowed. However, our motivation for allowing preemption comes from the fact that it is allowed in many application domains (such as scheduling in cloud computing) and has been widely considered in the broader scheduling literature.

Processing Functions. Following [17, 22], we consider processing functions that are (1) *non-decreasing* (assigning more machines does not decrease the rate of processing) and (2) *concave*[3] (the processing rate obeys the law of diminishing marginal utility because of greater overhead in coordination, communication costs, etc. between the

---

[3] We note that it is optimal to process an arbitrary available job on all machines simultaneously if all processing functions are *convex*.

machines processing a job).[4] Note that concave processing functions generalize the classical PS problem ($p_j(z) = 1$ iff $z \geq 1$ for all jobs j).

Integrality of machines. The existing literature is divided between allowing fractional allocation of machines to jobs (e.g. [22, 23]) or enforcing integrality of machine allocations (e.g., [16, 17, 20]). Accordingly, the processing functions are defined on the entire interval $[0, m]$ or on the discrete values $\{0, 1, \ldots, m\}$. Since we allow job preemption, a fractional assignment of machines to jobs can be realized by a round robin schedule even if there is no inherent support in the application for jobs to share a machine. Therefore, if the processing function is defined only on an integral domain, we extend it to the continuous domain by linear interpolation between adjacent points. In the rest of the paper, we will assume that the processing functions $p_j(.)$ are defined on the continuous domain $[0, m]$ and fractional schedules are valid.

Our Results. Our main result is a $(2+\epsilon)$-approximation algorithm for the GPS problem for concave processing functions. Note that this matches the best known bounds for the PS problem.

Theorem 1.1. *For any $\epsilon > 0$, there is a deterministic algorithm* GPSALGO *for the GPS problem that has an approximation factor of $(2 + \epsilon)$ for concave processing functions.*

We note that if preemption is disallowed, then the best approximation ratio known is 3.29 due to Jansen and Zhang [17].

In practice, a particularly relevant set of processing functions are *power functions* (for examples of their practical importance, see, e.g., [22]), i.e., $p_j(z) = c_j \cdot z^\gamma$ for $c_j > 0$. We show that our algorithm is in fact *optimal* for this special case. (Note that (1) power functions do not generalize the PS problem and (2) while the multiplier c_j can depend on the job, the exponent γ in the power functions has to be universal for our analysis.)

Theorem 1.2. *For any $\epsilon > 0$,* GPSALGO *has an approximation factor of $(1 + \epsilon)$ if the processing functions are power functions.*

Our Techniques. It would be natural to try to extend the greedy approach of Graham's algorithm for the PS problem to our problem. The basic scheduling rule of Graham's algorithm is the following: if there is an idle machine and an available job that is currently not being processed, then schedule this job on the machine. Note that this algorithm is *online* in the sense that it can operate on an instance where a job is revealed only after it becomes available. We categorically refute the possibility of extending this greedy approach to the GPS problem by giving a polynomial lower bound on the approximation factor obtained by *any* online algorithm for the GPS problem.

Theorem 1.3. *No online algorithm for the* GPS *problem can have a sub-polynomial competitive ratio, even if all the processing functions are a fixed power function.*

---

[4] Other classes of processing functions have also been considered in the literature (see related work), but monotonicity and concavity are two basic qualitative features of processing functions in most applications.

Instead, we employ an LP rounding approach (following the work of Chudak and Shmoys [5] and Skutella [26]). In designing the LP relaxation, we introduce a variable x_{ja} denoting the duration for which job j is processed simultaneously by a machines. Then, the processing time for job j is $X_j = \sum_a x_{ja}$. The goal is then to minimize the makespan T subject to the following constraints: (1) the total processing time of jobs on any chain (maximal directed path in the precedence graph) $\sum_{j \in C} X_j$ is a lower bound on the makespan T; (2) the total number of machine-hours for all jobs $\sum_j \sum_a a \cdot x_{ja}$ is a lower bound on mT; and (3) the total processing volume for any single job $\sum_a p_j(a) \cdot x_{ja}$ is at least its size s_j.

First, we solve our LP to obtain optimal values of x_{ja}'s. Next, we structure this solution by showing that $x_{ja} = 0$ for all *except one* value of a for each job j in an optimal solution w.l.o.g. Let us call this value b_j^*. We now create a feasible schedule by using the following simple rule: at any given time, we distribute the available jobs among all the m machines in proportion to their values of b_j^*. The key property that we use in the analysis is the following: (1) if there are too few available jobs (quantified by the sum of b_j^*'s of available jobs being less than m), then the non-decreasing property of the processing function ensures that for every chain, at least one job is being processed faster than in the LP solution, and (2) if there are too many available jobs (quantified by the sum of b_j^*'s of available jobs being greater than m), then the concavity of the processing function ensures that remaining overall ratio of the job volumes and machine-hours is decreasing at a faster rate than in the optimal LP solution. These two observations lead to the conclusion that the makespan of the schedule is at most twice the LP objective.

For the class of power functions, i.e., $p_j(z) = c_j \cdot z^\gamma$ (we only consider $\gamma \in [0, 1]$ since otherwise, the function is convex for which we have already shown that there is a simple optimal algorithm) our LP is exact (up to a factor of $(1 + \epsilon)$ for any $\epsilon > 0$). The main insight is that the special structure of power functions allows us to employ simple linear algebraic inequalities to design a function that trades off the two cases above. More precisely, we show that the gains/losses made by the algorithm over the optimal LP solution for the processing time of chains are exactly compensated by the losses/gains made by it over the LP solution for the overall number of machine-hours in the two situations described above.

Related Work. The precedence-constrained scheduling problem with malleable jobs has a long history in approximation algorithms. Du and Leung [6] showed that the problem is NP-hard even for a small number of machines and gave optimal algorithms if the precedence graph has special structure. Turek *et al* [28] considered the problem of scheduling malleable tasks in the absence of precedence constraints and obtained approximation algorithms for both the preemptive and non-preemptive situations. In the presence of precedence constraints, several families of processing functions have been considered. Our model was originally suggested by Prasanna and Musicus [22–24] and subsequently used by Jansen and Zhang [17], who obtained as approximation factor of 3.29 for the non-preemptive version of our problem. In some papers, the concavity requirement is replaced by a weaker constraint that the size of the jobs increases as more machines are assigned to it [16, 20]. For this model, the best known approximation factor is 4.73 [16]. A third (more general) model that has been considered is that of arbitrary speed up curves. Here, the processing rate not only depends on the number of

assigned machines and the job being processed, but also on the stage of processing of a job [8,9]. Most of the literature in this model is geared toward minimizing the flow-time (rather than the makespan) (see e.g. [2,10,11]), including in the presence of precedence constraints [25]. For a detailed survey on scheduling parallelizable jobs, the reader is referred to [7].

Since the work of Graham, both upper bounds (particularly, the trailing $o(1)$ factor in the approximation ratio) (see, e.g., [12,18]) and lower bounds [19,27] for the PS problem have been extensively studied. Moreover, multiple variants of this problem have been considered. This includes optimizing for other metrics such as completion time (see, e.g., [1] and references contained therein), handling machines with non-identical speeds [3,5], dealing with online input (e.g., [15]), etc. For a more detailed history of precedence constrained scheduling, the reader is referred to the surveys of Graham *et al* [14] and Chen *et al* [4].

2   Linear Program

In this section, we give a linear programming relaxation for the problem. In the discrete case, when the optimal solution allocates only an integral number of machines to each jobs, we let $A = \{1, \ldots, m\}$. In the continuous case, when the number of machines can be any real number from $[0, m]$, We pick an $\varepsilon > 0$, and let $A = \{(1-\varepsilon)^k \in [0, m] : k \in \mathbb{Z}\}$. Now for every job $j \in J$ and every value $a \in A$, we introduce a variable x_{ja}. In the intended solution corresponding to the optimal solution of GPS, x_{ja} is equal to the amount of time at which the number of machines used by the job j is between $(1 - \varepsilon)a$ and a (in the discrete case, $\varepsilon = 0$). We let T be the makespan of the schedule. Our goal is to minimize T. We write two constraints on T that are satisfied in the optimal solution.

To write the first constraint, we consider an arbitrary chain of jobs C. All jobs $j \in C$ must be processed sequentially one after another. It takes at least $\sum_{a \in A} x_{ja}$ amount of time to finish job j. Thus, for every chain C,

$$T \geq \sum_{j \in C} \sum_{a \in A} x_{ja}. \tag{1}$$

To write the second constraint, we count the number of machine hours used by the optimal solution. On one hand, every job j uses at least $\sum_{a \in A}(1 - \varepsilon)ax_{ja}$ machine hours. So the total number of machine hours is lower bounded by $\sum_{j \in J} \sum_{a \in A}(1 - \varepsilon)ax_{ja}$. On the other hand, the number of machine hours is upper bounded by mT. So we have

$$mT \geq \sum_{j \in J} \sum_{a \in A}(1 - \varepsilon)ax_{ja}. \tag{2}$$

To simplify notation, we let $\tilde{T} = T/(1 - \varepsilon)$. We finally add constraint (5) saying that every job j is completed in the optimal solution. We obtain the following LP relaxation.

$$\text{minimize } \tilde{T} \quad \text{subject to}$$

$$\sum_{j \in C} \sum_{a \in A} x_{ja} \leq \tilde{T} \qquad\qquad \text{for every chain } C \qquad (3)$$

$$\sum_{j \in J} \sum_{a \in A} a\, x_{ja} \leq \tilde{T}m \qquad\qquad\qquad\qquad (4)$$

$$\sum_{a \in A} x_{ja}\, p_j(a) \geq s_j \qquad\qquad \text{for every job } j \in J \qquad (5)$$

$$x_{ja} \geq 0 \qquad\qquad \text{for all } j \in J,\ a \in A \qquad (6)$$

Since the LP solution corresponding to the optimal solution satisfies all the constraints of the linear program, we have $LP \leq OPT$, where LP is the cost of the optimal LP solution, and OPT is the cost of the optimal combinatorial solution.

This linear program has infinitely many variables and exponentially many constraints, but using standard methods we can solve this linear program up to any precision $(1 + \varepsilon')$ in polynomial-time. For details, the reader is referred to the full version of the paper [21].

3   Simplified LP Solution

It turns out, that every solution to the LP (3–6) can be converted to another simpler solution in which for every job j one and only one x_{ja} is nonzero. Suppose x_{ja}^* is the optimal solution to the LP (3–6), define y_j^*'s and b_j^*'s as follows:

$$y_j^* = \sum_{a \in A} x_{ja}^*; \quad b_j^* = \frac{1}{y_j^*} \sum_{a \in A} x_{ja}^* a. \qquad (7)$$

Claim 1 *Variables y_j^* and b_j^* satisfy the following constraints (similar to (3–5)):*

$$\sum_{j \in C} y_j^* \leq \tilde{T} \qquad\qquad \text{for every chain } C \qquad (3')$$

$$\sum_{j \in J} b_j^* y_j^* \leq \tilde{T}m \qquad\qquad\qquad\qquad (4')$$

$$y_j^*\, p_j(b_j^*) \geq s_j \qquad\qquad \text{for every job } j \in J \qquad (5')$$

Proof. For every chain C, we have $\sum_{j \in C} y_j^* = \sum_{j \in C} \sum_{a \in A} x_{ja}^* \leq \tilde{T}$. Then,

$$\sum_{j \in J} b_j^* y_j^* = \sum_{j \in J} y_j^* \cdot \frac{1}{y_j^*} \sum_{a \in A} x_{ja}^* = \sum_{j \in J} \sum_{a \in A} x_{ja}^* \leq \tilde{T}m.$$

Finally, for every j, we have $y_j^*\, p_j(b_j^*) = y_j^*\, p_j\left(\frac{\sum_{a \in A} x_{ja}^* a}{\sum_{a \in A} x_{ja}^*}\right)$. Let $\lambda_{ja} = x_{ja}^* \big/ \sum_{a \in A} x_{ja}^*$. Then, $\sum_{a \in A} \lambda_{ja} = 1$ for every j. From concavity of the function $p_j(\cdot)$, we have

$$y_j^*\, p_j(b_j^*) = y_j^*\, p_j\left(\sum_{a \in A} \lambda_{ja} a\right) \geq y_j^* \sum_{a \in A} \lambda_{ja} p_j(a) = \sum_{a \in A} x_{ja}^* p_j(a) \geq s_j.$$

We can further assume that all constraints (5′) are tight i.e., for every j, we have $y_j^* \, p_j(b_j^*) = s_j$. Indeed, if (5′) is not tight for some j, then we can decrease y_j^* by letting $y_j^* = s_j / p_j(b_j^*)$.

4   Algorithm

We now describe the approximation algorithm. We first solve the LP relaxation and obtain a solution x_{ja}^*. Using Claim 1, we convert this solution to the solution (y_j^*, b_j^*) of the simplified LP (3′-5′). We assume that all constraints (5′) are tight (see above). Then we start the "rounding" procedure.

We schedule jobs iteratively. In every iteration, we schedule the next batch of jobs in the interval $[t, t + \Delta t]$ and then advance time from t to $t + \Delta t$. Thus, at the beginning of every iteration, we already have a schedule for the time interval $[0, t]$. For every job j, we keep the remaining size of j in the variable $s_j^*(t)$. Initially, $s_j^*(0) = s_j$. We also update the LP solution: we maintain variables $y_j^*(t)$ that indicate the time required by the remaining portion of job j if b_j^* machines are allotted to it. In other words, we maintain the invariant:

$$y_j^*(t) \cdot p_j(b_j^*) = s_j^*(t). \tag{8}$$

Initially, $y_j^*(0) = y_j^*$. Hence, for $t = 0$, this invariant holds.

To schedule the next batch of jobs, we find all unfinished jobs that can be scheduled now without violating precedence constraints. We call these *available* jobs. We denote the set of all available jobs at time t by $\Lambda(t)$. For every available job $j \in \Lambda(t)$ we compute

$$m_j^*(t) = m \cdot \frac{b_j^*}{\sum_{j \in \Lambda(t)} b_j^*}.$$

We allocate $m_j^*(t)$ machines to job j for the time interval of length

$$\Delta t = \min_{j \in \Lambda(t)} \frac{s_j^*(t)}{p_j(m_j^*(t))}.$$

Observe that the total number of machines we allocate is m. For all $j \in \Lambda(j)$, we update $s_j^*(t + \Delta t)$ and $y_j^*(t + \Delta t)$:

$$s_j^*(t + \Delta) = s_j^*(t) - p_j(m_j^*(t)) \, \Delta t \tag{9}$$

$$y_j^*(t + \Delta t) = y_j^*(t) - \frac{p_j(m_j^*(t))}{p_j(b_j^*)} \, \Delta t. \tag{10}$$

Note that this maintains invariant (8). We set $t = t + \Delta t$ and proceed to the next iteration. The algorithm terminates when $s_j^*(t) = 0$ for all j.

5   Analysis

We now analyze the algorithm. First, observe that the algorithm correctly maintains the remaining sizes $s_j^*(t)$: at time t, the remaining size of the job j is indeed $s_j^*(t)$.

Note that all $s_j^*(t)$ remain nonnegative (that is how we pick Δt). Moreover, at the end of every iteration one of the available jobs, specifically, the job j' for which $\Delta t = s_{j'}^*(t)/p_{j'}(m_{j'}^*(t))$ (again see the definition of Δt), is completed, i.e., $s_{j'}^*(t + \Delta t) = 0$. So the number of iterations of the algorithm is at most n, and the running time of the algorithm is polynomial in n. Also, note that all $y_j^*(t)$ are nonnegative by (8).

We now need to upper bound the makespan of the schedule produced by the algorithm. We prove the following standard lemma.

Lemma 5.1. *There exists a chain of jobs C^* such that at every point of time t one and only one job from C^* is scheduled by the algorithm.*

Proof. Consider the job j that finished last in the schedule generated by the algorithm. We add this job to our chain. This job was not scheduled earlier because it depends on some other job j' that finished just before j started. We add j' to our schedule as well. We then pick the job j' depends on, and so on. We continue this process until we encounter a job that does not depend on any other job. This job started at time $t = 0$. Thus, the jobs in the constructed chain cover the time line from the beginning to the end of the schedule.

We now show that for every t, the following inequality holds,

$$\sum_{j \in C^*} y_j^*(t) + \frac{1}{m} \sum_{j \in J} b_j^* y_j^*(t) \leq 2\tilde{T} - t. \tag{11}$$

Note that for $t = 0$, the inequality follows from (3') and (4'). This inequality implies that the makespan is at most $2\tilde{T} \leq 2(1 + \varepsilon)T$, since all $y_j^*(t)$ are nonnegative and thus the left hand side of (11) is nonnegative.

Lemma 5.2. *Inequality (11) holds in the beginning and end of every iteration.*

Proof. We assume that (11) holds at time t at the beginning of some iteration and prove that (11) holds at time $t + \Delta t$ at the end of this iteration. In an iteration, the RHS of (11) decreases by Δt. Our goal is to show that one of the following happens in any iteration:

- **Condition (a):** $\sum_{j \in C^*} y_j^*(t)$ (the first term in the LHS of (11)) decreases by at least Δt, or
- **Condition (b):** $\frac{1}{m} \sum_{j \in J} b_j^* y_j^*(t)$ (the second term in the LHS of (11)) decrease by at least Δt.

Since both the terms in the LHS of (11) are non-increasing, the lemma follows.

By Lemma 5.1, the algorithm schedules exactly one job in the chain C^* in the time interval $[t, t + \Delta t]$. We denote this job by j'. By equation (10), we have

$$y_{j'}^*(t + \Delta t) = y_{j'}^*(t) - \frac{p_{j'}(m_{j'}^*(t))}{p_{j'}(b_{j'}^*)} \Delta t = y_{j'}^*(t) - \frac{p_{j'}\left(\frac{m\, b_{j'}^*}{\sum_{j \in \Lambda(t)} b_j^*}\right)}{p_{j'}(b_{j'}^*)} \Delta t.$$

Denote $\alpha(t) = \sum_{j \in \Lambda(t)} b_j^*$. Rewrite the expression above as follows:

$$y_{j'}^*(t + \Delta t) = y_{j'}^*(t) - \frac{p_{j'}\left(b_{j'}^* \cdot m/\alpha(t)\right)}{p_{j'}(b_{j'}^*)} \Delta t. \tag{12}$$

Case 1: $\alpha(t) \leq m$: Since $p_{j'}(\cdot)$ is a non-decreasing function and $\alpha(t) \leq m$, we have $p_{j'}\big(b_{j'}^* \cdot m/\alpha(t)\big) \geq p_{j'}(b_{j'}^*)$. Using this fact in (12), we have

$$\sum_{j \in C^*} y_j^*(t + \Delta t) \leq \sum_{j \in C^*} y_j^*(t) - \Delta t.$$

Therefore, condition (a) holds in this case.

Case 2: $\alpha(t) \geq m$: We estimate the second term in the LHS of (11). Using (10), we have

$$\frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* y_j^*(t + \Delta t) = \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* \left(y_j^*(t) - \frac{p_j\big(b_j^* \, m/\alpha(t)\big)}{p_j(b_j^*)} \Delta t \right)$$

$$= \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* y_j^*(t) - \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* \frac{p_j\big(b_j^* \, m/\alpha(t)\big)}{p_j(b_j^*)} \Delta t.$$

Since $\alpha(t) \geq m$, we have $p_j\big(b_j^* \, m/\alpha(t)\big) \geq (m/\alpha(t)) \cdot p_j(b_j^*)$, since $p_j(\cdot)$ is a concave function with $p_j(0) = 0$. Thus,

$$\frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* y_j^* \cdot (t + \Delta t) \leq \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* y_j^*(t) - \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* \frac{m p_j(b_j^*)}{\alpha(t) p_j(b_j^*)} \Delta t$$

$$= \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* y_j^*(t) - \sum_{j \in \Lambda(t)} \frac{b_j^*}{\alpha(t)} \Delta t = \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* y_j^*(t) - \Delta t,$$

where the last equation follows from the definition of $\alpha(t)$. Therefore, condition (b) holds in this case.

Combining the two cases, no matter whether $\alpha(t) \leq m$ or $\alpha(t) \geq m$,

$$\sum_{j \in C^*} y_j^*(t + \Delta t) + \frac{1}{m} \sum_{j \in J} b_j^* y_j^*(t + \Delta t) \leq \sum_{j \in C^*} y_j^*(t) + \frac{1}{m} \sum_{j \in J} b_j^* y_j^*(t) - \Delta t.$$

This completes the proof.

6   Analysis for Power Functions

We now analyze the algorithm for power functions, i.e., $p_j(z) = c_j \cdot z^\gamma$ for some $\gamma \leq 1$ and constants $c_j > 0$. Let $\delta = 1 - \gamma$. We now show that for every t, the following inequality holds:

$$\left(\sum_{j \in C^*} y_j^*(t) \right)^\delta \cdot \left(\frac{1}{m} \sum_{j \in J} b_j^* y_j^*(t) \right)^\gamma \leq \tilde{T} - t. \tag{13}$$

Note that for $t = 0$, the inequality follows from (3′) and (4′). Inequality (13) implies that the makespan is at most $\tilde{T}$: all $y_j^*(t)$ are nonnegative, hence the left hand side of inequality (13) is also nonnegative, consequently $t \leq \tilde{T}$. Our main technical tool will be the following fact, which is an easy consequence of Hölder's inequality.

Fact 1 *Suppose $X, Y \geq 0$ and $\gamma, \delta \in [0, 1]$ such that $\gamma + \delta = 1$. For any $\Delta X \in [0, X]$, $\Delta Y \in [0, Y]$, define $\Delta \left(X^\delta \cdot Y^\gamma \right) = X^\delta \cdot Y^\gamma - (X - \Delta X)^\delta \cdot (Y - \Delta Y)^\gamma$. Then,*

$$\Delta \left(X^\delta \cdot Y^\gamma \right) \geq (\Delta X)^\delta \cdot (\Delta Y)^\gamma.$$

Proof. Define vectors $\mathbf{f} = ((X - \Delta X)^\delta, (\Delta X)^\delta)$ and $\mathbf{g} = ((Y - \Delta Y)^\gamma, (\Delta Y)^\gamma)$. Then, by Hölder's inequality, we have

$$\langle \mathbf{f}, \mathbf{g} \rangle \leq \|\mathbf{f}\|_{1/\delta} \|\mathbf{g}\|_{1/\gamma} \Rightarrow \quad (X - \Delta X)^\delta \cdot (Y - \Delta Y)^\gamma + (\Delta X)^\delta \cdot (\Delta Y)^\gamma \leq X^\delta \cdot Y^\gamma.$$

The lemma follows by rearranging terms.

Using this fact, we inductively prove that inequality (13) holds throughout the rounding algorithm.

Lemma 6.1. *Inequality (13) holds at the beginning and end of every iteration.*

Proof. We assume that (11) holds at time t at the beginning of some iteration and prove that (11) holds at time $t + \Delta t$ at the end of this iteration. By Lemma 5.1, the algorithm schedules exactly one job in the chain C^* in the time interval $[t, t + \Delta t]$. We denote this job by j'. We have

$$y_{j'}^*(t + \Delta t) = y_{j'}^*(t) - \frac{p_{j'}(m_{j'}^*(t))}{p_{j'}(b_{j'}^*)} \Delta t = y_{j'}^*(t) - \frac{p_{j'}\left(\frac{m\, b_{j'}^*}{\sum_{j \in \Lambda(t)} b_j^*} \right)}{p_{j'}(b_{j'}^*)} \Delta t.$$

Denote $\alpha(t) = \sum_{j \in \Lambda(t)} b_j^*$. Rewrite the expression above as follows:

$$y_{j'}^*(t + \Delta t) = y_{j'}^*(t) - \frac{p_{j'}\left(b_{j'}^* \cdot m/\alpha(t) \right)}{p_{j'}(b_{j'}^*)} \Delta t = y_{j'}^*(t) - \left(\frac{m}{\alpha(t)} \right)^\gamma \Delta t.$$

We estimate the second term in (13).

$$\frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* y_j^* \cdot (t + \Delta t) = \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* \left(y_j^*(t) - \frac{p_j\left(b_j^*\, m/\alpha(t) \right)}{p_j(b_j^*)} \Delta t \right)$$

$$= \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* y_j^*(t) - \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* \frac{p_j\left(b_j^*\, m/\alpha(t) \right)}{p_j(b_j^*)} \Delta t = \frac{1}{m} \sum_{j \in \Lambda(t)} b_j^* y_j^*(t) - \left(\frac{\alpha(t)}{m} \right)^\delta \Delta t.$$

Using Fact 1, we have

$$\Delta \left(\left(\sum_{j \in C^*} y_j^*(t) \right)^\delta \cdot \left(\frac{1}{m} \sum_{j \in J} b_j^* y_j^*(t) \right)^\gamma \right) \geq \left(\Delta \left(\sum_{j \in C^*} y_j^*(t) \right) \right)^\delta \cdot \left(\Delta \left(\frac{1}{m} \sum_{j \in J} b_j^* y_j^*(t) \right) \right)^\gamma$$

$$= \left(\left(\frac{m}{\alpha(t)} \right)^\gamma \Delta t \right)^\delta \cdot \left(\left(\frac{\alpha(t)}{m} \right)^\delta \Delta t \right)^\gamma = \Delta t.$$

Note that when we move from time t to time $t + \Delta t$, the right hand side of (11) decreases by Δt. This completes the proof.

7   Lower Bound for Online Algorithms

In this section, we show that the GPS problem has a polynomial lower bound in the online setting. Specifically, we show that even if $p_j(z) = \sqrt{z}$ for all jobs j, the competitive ratio of any online algorithm is at least $\Omega(n^{1/4})$, where n is the number of jobs. Note that in this case our offline algorithm gives an almost exact solution ($(1 + \varepsilon)$ approximation for arbitrary $\varepsilon > 0$). In the online model, a job is given to the algorithm only once all the jobs it depends on are finished. So this result shows that any approximation algorithm should use the information about the future schedule and cannot make the decision solely based on the set of currently available jobs.

We describe the strategy for the adversary. The adversary works in phases. In phase i, she presents l independent jobs $u_{i1}, \ldots, u_{il}$ to the algorithm. These jobs depend on the single job in the previous phase that has finished last. That is, if u_{ij_i} is the job that finished last in the phase i, then in the next phase $(i + 1)$, all jobs $u_{(i+1)s}$ depend on this job u_{ij_i}. We assume that the number of machines is $m = 1$.

We lower bound the makespan of the schedule produced by the online algorithm. To finish all l jobs given to the algorithm in one phase, we need to spend time at least $\sqrt{l}$. (The optimal way to allocate machines is to assign $1/l$ machines to each job.) Thus, the total length of the schedule is at least $k\sqrt{l}$.

Now consider the following solution. Initially, all jobs in the chain $v_{1j_1}, v_{2j_2}, \ldots, v_{kj_k}$ are scheduled sequentially (assigning 1 machine to each job). Once all jobs v_{ij_i} are finished, the remaining $kl - k$ jobs are scheduled in parallel by assigning $1/(kl - l)$ machines to every job. The length of the schedule equals $k + \sqrt{kl - k}$. The lower bound now follows by setting $k = l$ (note that $n = kl$). This lower bound can be extended to randomized algorithms using standard techniques, which we omit for brevity.

We note that this lower bound is almost tight for $p_j(z) = \sqrt{z}$: any algorithm that does not idle (i.e., which always allocates all available machines) has competitive ratio at most $\sqrt{n}$, since the maximum possible rate of processing all jobs is $\sqrt{n}$ (when we process all n jobs in parallel) and the minimum possible rate is 1 (when we allocate all machines to a single job).

References

1. Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *FOCS*, pages 453–462, 2009.
2. Ho-Leung Chan, Jeff Edmonds, and Kirk Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. *Theory Comput. Syst.*, 49(4):817–833, 2011.
3. Chandra Chekuri and Michael A. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *J. Algorithms*, 41(2):212–224, 2001.
4. B. Chen, C.N. Potts, and G.J. Woeginger. A review of machine scheduling: Complexity, algorithms and approximability. *Handbook of Combinatorial Optimization*, 3:21–169, 1998.
5. Fabián A. Chudak and David B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *J. Algorithms*, 30(2):323–343, 1999.
6. Jianzhong Du and Joseph Y.-T. Leung. Scheduling tree-structured tasks on two processors to minimize schedule length. *SIAM J. Discrete Math.*, 2(2):176–196, 1989.

7. P.-F. Dutot, G. Mounie, and D. Trystram. Scheduling parallel tasks approximation algorithms. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 2004.
8. Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.
9. Jeff Edmonds, Donald D. Chinn, Tim Brecht, and Xiaotie Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. *J. Scheduling*, 6(3):231–250, 2003.
10. Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. *ACM Transactions on Algorithms*, 8(3):28, 2012.
11. Kyle Fox, Sungjin Im, and Benjamin Moseley. Energy efficient scheduling of parallelizable jobs. In *SODA*, pages 948–957, 2013.
12. Devdatta Gangal and Abhiram G. Ranade. Precedence constrained scheduling in (2 - 7/(3p+1)) optimal. *J. Comput. Syst. Sci.*, 74(7):1139–1146, 2008.
13. R. L. Graham. Bounds for certain multiprocessing anomalies. *Siam Journal on Applied Mathematics*, 1966.
14. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5(2):287–326, 1979.
15. Yumei Huo and Joseph Y.-T. Leung. Online scheduling of precedence constrained tasks. *SIAM J. Comput.*, 34(3):743–762, 2005.
16. Klaus Jansen and Hu Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Transactions on Algorithms*, 2(3):416–434, 2006.
17. Klaus Jansen and Hu Zhang. Scheduling malleable tasks with precedence constraints. *J. Comput. Syst. Sci.*, 78(1):245–259, 2012.
18. Shui Lam and Ravi Sethi. Worst case analysis of two scheduling algorithms. *SIAM J. Comput.*, 6(3):518–536, 1977.
19. J. K. Lenstra and Rinnooy A. H. G. Kan. Complexity of Scheduling under Precedence Constraints. *Operations Research*, 26(1):22–35, 1978.
20. Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. *Int. J. Found. Comput. Sci.*, 13(4):613–627, 2002.
21. Konstantin Makarychev and Debmalya Panigrahi. Precedence-constrained scheduling of malleable jobs with preemption. *CoRR*, abs/1404.6850, 2014.
22. G. N. Srinivasa Prasanna and Bruce R. Musicus. Generalised multiprocessor scheduling using optimal control. In *SPAA*, pages 216–228, 1991.
23. G. N. Srinivasa Prasanna and Bruce R. Musicus. Generalized multiprocessor scheduling for directed acyclic graphs. In *SC*, pages 237–246, 1994.
24. G. N. Srinivasa Prasanna and Bruce R. Musicus. The optimal control approach to generalized multiprocessor scheduling. *Algorithmica*, 15(1):17–49, 1996.
25. Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *SODA*, pages 491–500, 2008.
26. Martin Skutella. Approximation algorithms for the discrete time-cost tradeoff problem. In *SODA*, pages 501–508, 1997.
27. Ola Svensson. Hardness of precedence constrained scheduling on identical machines. *SIAM J. Comput.*, 40(5):1258–1274, 2011.
28. John Turek, Joel L. Wolf, and Philip S. Yu. Approximate algorithms scheduling parallelizable tasks. In *SPAA*, pages 323–332, 1992.