Bag-Of-Tasks Scheduling on Related Machines

Anupam Gupta
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Amit Kumar
Computer Science and Engineering Department, Indian Institute of Technology, Delhi, India

Sahil Singla
Department of Computer Science, Princeton University, NJ, USA

Abstract

We consider online scheduling to minimize weighted completion time on related machines, where each *job* consists of several *tasks* that can be concurrently executed. A job gets completed when all its component tasks finish. We obtain an $O(K^3 \log^2 K)$-competitive algorithm in the *non-clairvoyant* setting, where K denotes the number of distinct machine speeds. The analysis is based on dual-fitting on a precedence-constrained LP relaxation that may be of independent interest.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases approximation algorithms, scheduling, bag-of-tasks, related machines

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2021.3

Category APPROX

Related Version *Full Version*: https://arxiv.org/abs/2107.06216

Funding *Anupam Gupta*: Supported in part by NSF awards CCF-1907820, CCF1955785, and CCF-2006953.

1 Introduction

Scheduling to minimize the weighted completion time is a fundamental problem in scheduling. Many algorithms have been developed in both the online and offline settings, and for the cases where machines are identical, related, or unrelated. Most of the work, however, focuses on the setting where each job is a monolithic entity, and has to be processed in a sequential manner.

In this work, we consider the online setting with multiple related machines, where each *job* consists of several *tasks*. These tasks are independent of each other, and can be executed concurrently on different machines. (Tasks can be preempted and migrated.) A job is said to have *completed* when all its component tasks finish processing. We consider the *non-clairvoyant* setting where the algorithm does not know the size of a task up-front, but only when the task finishes processing. Such instances arise in operating system schedulers, where a job and its tasks correspond to a process and its threads that can be executed in parallel. This setting is sometimes called a *"bag of tasks"* (see e.g. [2, 10, 4]).

The bag-of-tasks model can be modeled using precedence constraints. Indeed, each job is modeled as a star graph, where the tasks correspond to the leaves (and have zero weight), and the root is an auxiliary task with zero processing requirement but having weight w_j. Hence the root can be processed only after all leaf tasks have completed processing. The goal is to minimize total *weighted completion time*. Garg et al. [7] gave a constant-competitive algorithm for this problem for *identical machines*, in a more general setting where tasks form arbitrary precedence DAGs.

We extend this result to the setting of *related machines* where machine i has speed s_i. By losing a constant factor, we assume that all speeds are powers of some constant C. Let K denote the number of distinct machine speeds. In §2, we show that this problem is strictly more challenging than in the identical machines setting:

▶ **Theorem 1** (Lower Bound). *Any online non-clairvoyant algorithm has $\Omega(K)$ competitive ratio for bags-of-tasks on related machines.*

The lower bound arises because we want to process larger tasks on fast machines, but we have no idea about the sizes of the tasks, so we end up clogging the fast machines with small tasks: this issue did not arise when machines were identical. Given the lower bound, we now look for a non-clairvoyant scheduling algorithm with a competitive ratio that depends on K, the number of distinct speeds. This number may be small in many settings, e.g., when we use commodity hardware of a limited number of types (say, CPUs and GPUs). Our main result is a positive answer to this question:

▶ **Theorem 2** (Upper Bound). *The online non-clairvoyant algorithm for bags-of-tasks on related machines has a competitive ratio of §3 is $O(\min\{K^3 \log^2 K, K + \log n\})$.*

Our algorithm uses a greedy strategy. Instead of explicitly building a schedule, it assigns *(processing) rates* to tasks at each time t. Such a rate assignment is called feasible if for every k, the rate assigned to any subset of k tasks is at most the total speed of the k fastest machines. Using an argument based on Hall's matching theorem, a schedule exists if and only if such a rate assignment can be found. To assign these rates, each alive task gets a "priority", which is the ratio of the weight of the job containing it to the number of alive tasks of this job. In other words, a task with low weight or with many tasks gets a low priority. We assign feasible rates to alive tasks in a "fair manner", i.e., we cannot increase the rate of a high priority task by decreasing the rate of a lower priority task. To efficiently find such feasible rates, we use a water-filling procedure.

The analysis proceeds using the popular dual-fitting approach, but we need new ideas: **(i)** we adapt the precedence-constrained LP relaxation for completion time in [5] to our setting. A naive relaxation would define the completion time of a task as the maximum of the (fractional) completion times of each of the tasks, where the fractional completion time of a task is the sum over times t of the fraction of the task remaining at this time Instead, we define $U_{j,t}$, for a job j and time t as the maximum over all tasks v for j of the fraction of v which remains to be completed at time t, the completion time of j as $\sum_t U_{jt}$. (See §4 for details.) **(ii)** Although it is natural to divide the machines into classes based on their speeds, we need a finer partitioning, which drives our setting of dual variables. Indeed, the usual idea of dividing up the job's weight equally among the tasks that are still alive only leads to an $O(\log n)$-competitiveness (see §5). To do better, we first preprocess the instance so that distinct machine speeds differ by a constant factor, but the total processing capacity of a slower speed class is far more than that of all faster machines. Now, at each time, we divide the machines into *blocks*. A constant fraction of the blocks have the property that either the average speed of the machines in the block is close to one of the speed classes, or the total processing capacity of a block is close to that of *all* the machines of a speed class. It turns out that our dual-fitting approach works for accounting the weight of jobs which get processed by such blocks; proving this constitutes the bulk of technical part of the analysis. Finally, we show that most jobs (in terms of weight) get processed by such blocks, and hence we are able to bound the overall weighted completion time. We present the proofs in stages, giving intuition for the new components in each of the sections.

1.1 Related Work

Minimizing weighted completion time on parallel machines with precedence constraints has $O(1)$-approximation in the *offline* setting: Li [9] improves on [8, 11] to give a $3.387+\varepsilon$-approximation. For *related* machines the precedence constraints make the problem harder: there is an $O(\log m/\log\log m)$-approximation [9] improving on a prior $O(\log K)$ result [5], and an $\omega(1)$ hardness under certain complexity assumptions [3]. Here m denotes the number of machines. These results are for offline and hence clairvoyant settings, and do not apply to our setting of non-clairvoyant scheduling.

In the setting of parallel machines, there has been recent work on minimizing weighted completion time in DAG scheduling, where each job consists of a set of tasks with precedence constraints between them given by a DAG [13, 1]. [7] generalized this to the non-clairvoyant setting and gave an $O(1)$-competitive algorithm. Our algorithm for the related case is based on a similar water-filling rate assignment idea. Since the machines have different speeds, a set of rates assigned to tasks need to satisfy a more involved feasibility condition. Consequently, its analysis becomes much harder; this forms the main technical contribution of the paper. Indeed, even for the special case considered in this paper where every DAG is a star, we can show a lower bound of $\Omega(K)$ on the competitive ratio of any non-clairvoyant algorithm. In the full version, we show that any non-clairvoyant algorithm for related machines DAG scheduling must have $\Omega\big(\frac{\log m}{\log\log m}\big)$-competitive ratio.

Our problem also has similarities to open shop scheduling. In open shop scheduling, each jobs consists of several tasks, where each task v (for job j) needs to be processed on a distinct machine for p_{vj} amount of time. However, unlike our setting, two tasks for a job cannot be processed simultaneously on different machines. [12] considered open shop scheduling in the offline setting for related machines and gave a $(2+\varepsilon)$-approximation. [6] considered a further generalization of our problem to unrelated machines, where the tasks corresponding to distinct jobs need not be disjoint. They gave a constant-factor approximation algorithm, again offline.

1.2 Paper Organization

In this extended abstract, we first give the algorithm in §3, and the linear program in §4. A simpler proof of $O(K+\log n)$-competitiveness is in §5. We show poly(K)-competitiveness for the case of a single job (which corresponds to makespan minimization) in §6, and then give the complete proof for the general case in §7.

2 Problem Statement and the $\Omega(K)$ Hardness

Each *job* j has a *weight* w_j and consists of *tasks* $T(j)=\{(j,1),(j,2),\ldots,(j,k_j)\}$ for some k_j. Each task $v=(j,\ell)$ has an associated *processing requirement/size* $p_v=p_{(j,\ell)}$. The job j completes when all its associated tasks finish processing. We use letters j, j', etc. to denote jobs, and v, v', etc. to denote tasks (j,ℓ).

There are m machines with speeds $s_1\geq s_2\geq\ldots\geq s_m$. The goal is to minimize the weighted completion time of the jobs. We allow task *preemption* and *migration*, and different tasks of a job can be processed concurrently on different machines. However, a task itself can be processed on at most one machine at any time. In this extended abstract we consider the special case when all release dates are 0, but our results also extend to the more general setting of arbitrary release dates (details in the full version). Let $S_k:=s_1+\ldots+s_k$ denote the total speed of the fastest k machines. Since we care about the number of *distinct* speeds, we assume there are K *speed classes*, with speeds $\sigma_1>\sigma_2>\ldots>\sigma_K$. There are m_i machines having speed σ_i, where $\sum_i m_i=m$.

▶ **Assumption 3** (Increasing Capacity Assumption). *For parameter $\gamma \geq 1$:*

(1) (Falling Speeds.) *For each ℓ, we have $\sigma_i/\sigma_{i+1} \geq 64$.*

(2) (Increasing Capacity.) *For each ℓ, the total processing capacity of speed class ℓ is at least twice that of the previous (faster) speed classes. I.e., $m_\ell\sigma_\ell \geq 2(m_1\sigma_1 + \ldots + m_{\ell-1}\sigma_{\ell-1})$.*

(3) (Speed-up.) *The algorithm uses machines that are γ times faster than the adversary's machines.*

▶ **Proposition 4.** *An arbitrary instance can be transformed into one satisfying Assumption 3 by losing a factor $O(\gamma K)$ in the competitive ratio.*

Proof. *(Sketch)* For the first part, we round down the speed of each machine to a power of 64. This changes the completion time by at most a factor of 64. The second increasing capacity assumption is not without loss of generality – we greedily find a subset of speed classes by losing $O(K)$ factor in competitive ratio (see details in Appendix A). Finally, the γ-speedup can only change the competitive ratio by γ factor. ◀

Next we show that any online algorithm has to be $\Omega(K)$-competitive even for a single job with the machines satisfying increasing capacity Assumption 3.

▶ **Proposition 5.** *Any online algorithm is $\Omega(K)$-competitive even for a single job under increasing capacity Assumption 3.*

Proof. *(Sketch)* Consider a single job j with m tasks, where m is the number of machines. For every speed class ℓ, there are m_ℓ tasks of size σ_ℓ – call these tasks $T_\ell(j)$. Since there is only one job, the objective is to minimize the makespan. The offline (clairvoyant) objective is 1, since all tasks can be assigned to machines with matching speeds. However, any online algorithm incurs a makespan of $\Omega(K)$. Here is an informal argument, which can be proved even for randomized algorithms against oblivious adversaries: since there is no way to distinguish between the tasks, the algorithm can at best run all the alive tasks at the same speed. The tasks in $T_K(j)$ will be the first to finish by time $\frac{m_K\sigma_K}{\sum_\ell m_\ell\sigma_\ell} \geq \frac{1}{2}$, where the inequality follows from the increasing capacity assumption. At this time, the processing on tasks from $T_\ell(j)$ for $\ell < K$ has been very small, and so tasks in $T_{K-1}(j)$ will require about $1/2$ more units of time to finish, and so on. ◀

3 The Scheduling Algorithm

The scheduling algorithm assigns, at each time t, a *rate* L_v^t to each unfinished task v. The following lemma (whose proof is deferred to the appendix) characterizes rates that correspond to schedules:

▶ **Lemma 6.** *A schedule $\mathcal{S}$ is feasible if for every time t and every value of k:*

($\star$) *the total rate assigned to any subset of k tasks is at most $\gamma \cdot S_k$.*

For each time t, we now specify the rates L_v^t assigned to each unfinished task v. For job j, let $T^t(j)$ be the set of tasks in $T(j)$ which are alive at time t. Initially all tasks are *unfrozen*. We raise a parameter τ, starting at zero, at a uniform speed. The values taken by τ will be referred to as *moments*. For each job j and each task $v \in T^t(j)$ that is unfrozen, define a *tentative rate* at τ to be

$$L_v^t := \frac{w_j}{|T^t(j)|} \cdot \tau \ . \tag{1}$$

Hence the tentative rates of these unfrozen tasks increase linearly, as long as condition ($\star$) is satisfied. However, if ($\star$) becomes tight for some subset V of alive tasks, i.e., $\sum_{v \in V} L_v^t = \gamma \cdot S_{|V|}$, pick a *maximal* set of such tasks and *freeze* them, fixing their rates at their current tentative values. (Observe the factor of γ appears on the right side because we assume the machines in the algorithm to have a speedup of γ.) Now continue the algorithm this way, raising τ and the L_v^t values of remaining unfrozen tasks v until another subset gets tight, etc., stopping when all jobs are frozen. This defines the L_v^t rates for each task v for time t. By construction, these rates satisfy ($\star$).

3.1 Properties of the Rate Assignment

The following claim shows that all alive tasks corresponding to a job get frozen simultaneously.

▶ **Lemma 7** (Uniform Rates). *For any time t and any job j, all its alive tasks (i.e., those in $T^t(j)$) freeze at the same moment τ, and hence get the same rate.*

Proof. For the sake of contradiction, consider the first moment τ where a maximal set V of tasks contains v but not v', for some job j with $v, v' \in T(j)$. Both v, v' have been treated identically until now, so $L_v^t = L_{v'}^t$. Also, by the choice of τ, $\sum_{u \in V : u \neq v} L_u^t + L_v^t = \gamma S_{|V|}$. Since we maintain feasibility at all moments,

$$\sum_{u \in V : u \neq v} L_u^t + L_v^t + L_{v'}^t \leq \gamma S_{|V|+1} \qquad \text{and} \qquad \sum_{u \in V : u \neq v} L_u^t \leq \gamma S_{|V|-1} \ .$$

This implies $L_v^t \geq \gamma s_{|V|}$ and $L_{v'}^t \leq \gamma s_{|V|+1}$. Since $L_v^t = L_{v'}^t$ and $s_{|V|} \geq s_{|V|+1}$, all of these must be equal. In that case, by the maximality of set V, the algorithm should have picked $V \cup \{v'\}$ instead of V. ◀

For a task $v \in T^t(j)$, define $\widetilde{w}^t(v) := w_j / |T^t(j)|$ to be task v's "share" of the weight of job j at time t. So if task v freezes at moment τ, then its rate is $L_v^t = \widetilde{w}^t(v) \cdot \tau$. Let us relate this share for v to certain averages of the weight. (Proof in Appendix B)

▶ **Corollary 8.** *Fix a time t. Let V be the set of tasks frozen by some moment τ. For a task $v \in V$,*

- **(i)** *if $V' \subseteq V$ is any subset of tasks which freeze either at the same moment as v, or after it, then $\frac{\widetilde{w}^t(v)}{s_{|V|}} \geq \frac{w(V')}{S_{|V|}}$.*
- **(ii)** *if $V'' \subseteq V$ is any subset of tasks which freeze either at the same moment as v, or before it, then $\frac{\widetilde{w}^t(v)}{L_v^t} \leq \frac{\widetilde{w}^t(V'')}{\sum_{v' \in V''} L_{v'}^t}$.*

3.2 Defining the Blocks

The rates for tasks alive at any time t are defined by a sequence of freezing steps, where some group of tasks are frozen: we call these groups *blocks*. By Lemma 7, all tasks in $T^t(j)$ belong to the same block. The weight $w(B)$ of block B is the total weight of jobs whose tasks belong to B. Let $B_1^t, B_2^t, \ldots$ be the blocks at time t in the order they were frozen, and $\tau_1^t, \tau_2^t, \ldots$ be the moments at which they froze. Letting $b_r^t := |B_1^t \cup \ldots \cup B_r^t|$, we get that any task $v \in B_r^t$ satisfies $\tau_v^t \cdot w(B_r^t) = \gamma(S_{b_{r+1}^t} - S_{b_r^t})$.

Each block B_r^t has an associated set of machines, namely the machines on which the tasks in this block are processed – i.e., the machines indexed $b_{r-1}^t + 1, \ldots, b_r^t$. We use $m(B)$ to denote the set of machines associated with a block B. Since $|B| = |m(B)|$ and the jobs in B are processed on $m(B)$ in a pre-emptive manner at time t, the rate assigned to any job is at least the slowest speed (and at most the fastest speed) of the machines in $m(B)$.

4 The Analysis and Intuition

We prove the competitiveness by a dual-fitting analysis: we give a primal-dual pair of LPs, use the algorithm above to give a feasible primal, and then exhibit a feasible dual with value within a small factor of the primal cost.

In the primal LP, we have variables x_{ivt} for each task v, machine i, and time t denoting the extent of processing done on task v at machine i during the interval $[t, t+1]$. Here $U_{j,t}$ denotes fraction of job j finished at or after time t, and C_j denotes the completion time of job j.

$$\begin{aligned}
&\min \textstyle\sum_j w_j C_j + \sum_{j,t} w_j U_{j,t} && && \\
U_{j,t} &\geq \textstyle\sum_{t' \geq t} \sum_i \frac{x_{ivt'}}{p_v} && \forall j, \forall v \in T(j), \forall t && (2)\\
C_j &\geq \textstyle\sum_t \sum_i \frac{x_{ivt}}{s_i} && \forall j, \forall v \in T(j) && (3)\\
\textstyle\sum_i \sum_t \frac{x_{ivt}}{p_v} &\geq 1 && \forall j, \forall v \in T(j) && (4)\\
\textstyle\sum_v \frac{x_{ivt}}{s_i} &\leq 1 && \forall i, \forall t && (5)
\end{aligned}$$

The constraint (2) is based on precedence-constrained LP relaxations for completion time. Indeed, each job can be thought of as a star graph with a zero size task at the root preceded by all the actual tasks at the leaf. In our LP, for each time t, we define $U_{j,t} \in [0,1]$ to be the maximum over all tasks $v \in T(j)$ of the fraction of v that remains (the RHS of (2)), and the completion time of j is at least the total sum over times t of $U_{j,t}$ values. Since we do not explicitly enforce that a task cannot be processed simultaneously on many machines, the first term $\sum_j w_j C_j$ is added to avoid a large integrality gap. We show feasibility of this LP relaxation (up to factor 2) in §C.

▷ Claim 9. For any schedule $\mathcal{S}$, there is a feasible solution to the LP of objective value at most $2\,\mathsf{cost}(\mathcal{S})$.

The linear programming dual has variables $\alpha_{j,v}, \delta_{j,v}, \delta_{j,v,t}$ corresponding to constraints (4),(3),(2) for every job j and task $v \in T(j)$, and $\beta_{i,t}$ corresponding to constraints (5) for every machine i and time t:

$$\begin{aligned}
&\max . \textstyle\sum_{j,v} \alpha_{j,v} - \sum_{i,t} \beta_{i,t} && && \\
\tfrac{\alpha_{j,v}}{p_v} &\leq \tfrac{\beta_{i,t}}{s_i} + \textstyle\sum_{t' \leq t} \frac{\delta_{j,v,t}}{p_v} + \frac{\delta_{j,v}}{s_i} && \forall j, \forall i, \forall t, \forall v \in T(j) && (6)\\
\textstyle\sum_{v \in T(j)} \delta_{j,v} &\leq w_j && \forall j && (7)\\
\textstyle\sum_{v \in T(j)} \delta_{j,v,t} &\leq w_j && \forall j, t && (8)
\end{aligned}$$

We now give some intuition about these dual variables. The quantity $\delta_{j,v,t}$ should be thought of the contribution (at time t) towards the weighted flow-time of j. Similarly, $\delta_{j,v}$ is *global* contribution of v towards the flow-time of v. (In the integral case, $\delta_{j,v}$ would be w_j for the task which finishes last. If there are several such tasks, $\delta_{j,v}$ would be non-zero only for such tasks only and would add up to w_j). The quantity $\alpha_{j,v}$ can be thought of as v's contribution towards the total weighted flow-time, and $\beta_{i,t}$ is roughly the queue size at time t on machine i. Constraint (6) upper bounds $\alpha_{j,v}$ in terms of the other dual variables. More intuition about these variables can be found in §4.2.

4.1 Simplifying the dual LP

Before interpreting the dual variables, we rewrite the dual LP and add some additional constraints. Define additional variables $\alpha_{j,v,t}$ for each job j and task $v \in T(j)$ and time t, such that variable $\alpha_{j,v} = \sum_t \alpha_{j,v,t}$. We add a new constraint:

$$\sum_{v \in T(j)} \alpha_{j,v,t} \leq w_j. \tag{9}$$

This condition is not a requirement in the dual LP, but we will set $\alpha_{j,v,t}$ to satisfy it. Assuming this, we set $\delta_{j,v,t} := \alpha_{j,v,t}$ for all jobs j, tasks $v \in T(j)$ and times t; feasibility of (9) implies that of (8). Moreover, (6) simplifies to

$$\sum_{t' \geq t} \frac{\alpha_{j,v,t'}}{p_v} \leq \frac{\beta_{i,t}}{s_i} + \frac{\delta_{j,v}}{s_i}.$$

Observe that we can write p_v as the sum of the rates, and hence as $p_v = \sum_{t'} L_v^{t'}$. Since this is at least $\sum_{t' \geq t} L_v^{t'}$ for any t, we can substitute above, and infer that it suffices to verify the following condition for all tasks $v \in T(j)$, time t, and time $t' \geq t$:

$$\alpha_{j,v,t'} \leq \frac{\beta_{i,t} \cdot L_v^{t'}}{s_i} + \frac{\delta_{j,v} \cdot L_v^{t'}}{s_i}. \tag{10}$$

Henceforth, we ensure that our duals (including $\alpha_{j,v,t}$) satisfy (9),(10) and (7).

4.2 Interpreting the Duals and the High-Level Proof Idea

We give some intuition about the dual variables, which will be useful for understanding the subsequent analysis. We set dual variables $\alpha_{j,v}$ such that for any job j, the sum $\sum_{v \in T(j)} \alpha_{j,v}$ is (approximately) the weighted completion of job j. This ensures that $\sum_{j,v} \alpha_{j,v}$ is the total weighted completion of the jobs. One way of achieving this is as follows: for every time t and task-job pair (j, v) we define $\alpha_{j,v,t}$ variables such that they add up to be w_j if job j is unfinished at time t (i.e., (9) is satisfied with equality). If $\alpha_{j,v}$ is set to $\sum_t \alpha_{j,v,t}$, then these $\alpha_{j,v}$ variables would add up to the weighted completion time of j.

The natural way of defining $\alpha_{j,v,t}$ is to evenly distribute the weight of j among all the alive tasks at time t, i.e., to set $\alpha_{j,v,t} = \frac{w_j}{T^t(j)}$. This idea works if we only want to show that the algorithm is $O(\log n)$-competitive, but does not seem to generalize if we want to show $O(K)$-competitiveness. The reason for this will be clearer shortly, when we discuss the $\delta_{j,v}$ variables.

Now we discuss $\beta_{i,t}$ dual variables. We set these variables so that $\sum_t \beta_{i,t}$ is a constant (less than 1) times the total weighted completion time. This ensures that the objective value of the dual LP is also a constant times the total weighted completion time. A natural idea (ignoring constant factors for now) is to set $\beta_{i,t} = \frac{w(A^t)}{K m_\ell}$, where A^t is the set of alive jobs at time t and ℓ is the speed class of machine i. Since we have put an $\Omega(K)$ term in the denominator of $\beta_{i,t}$ (and no such term in the definition of $\alpha_{j,v}$), ensuring the feasibility of (6) would require a speed augmentation of $\Omega(K)$.

Finally, consider the $\delta_{j,v}$ dual variables. As (7) suggests, setting $\delta_{j,v}$ is the same as deciding how to distribute the weight w_j among the tasks in $T(j)$. Notice, however, that this distribution cannot depend on time (unlike $\alpha_{j,v,t}$ where we were distributing w_j among all the alive tasks at time t). In the ideal scenario, tasks finishing later should get high $\delta_{j,v}$ values. Since we are in the non-clairvoyant setting, we may want to set $\delta_{j,v} = \frac{w_j}{|T(j)|}$. We now argue this can lead to a problem in satisfying (10).

Consider the setting of a single unit-weight job j initially having n tasks, and so we set $\delta_{j,v} = \frac{1}{n}$ for all v. Say that $n = m_\ell$ for a large value of ℓ: by the increasing capacity assumption, $m_\ell \approx m_1 + \ldots + m_\ell$. Now consider a later point in time t when only n' tasks

remain, where $n' = m_{\ell'}$ for some speed class $\ell' \ll \ell$. At this time t, each of the n' surviving tasks have $\alpha_{j,v,t} = \frac{1}{n'}$. But look at the RHS of (10), with machine i of speed class ℓ. The rate L_v^t will be very close to $\sigma_{\ell'}$ (again, by the increasing capacity assumption), and so both the terms would be about $\frac{\sigma_{\ell'}}{m_\ell \sigma_\ell}$. However, $m_\ell \sigma_\ell$ could be much larger than $m_{\ell'}\sigma_{\ell'}$, and so this constraint will not be satisfied. In fact, we can hope to satisfy (10) at some time t only if n' is close to n, say at least $n/2$. When the number of alive tasks drops below $n/2$, we need to *redistribute* the weight of j among these tasks, i.e., we need to increase the $\delta_{j,v}$ value for these tasks, to about $\frac{1}{n/2}$. Since these halving can happen for $\log n$ steps, we see that (3) is violated by a factor of $\log n$. These ideas can be extended to give an $O(\log n + K)$-competitive algorithm for arbitrary inputs; see §5 for details. To get a better bound, we need a more careful setting of the dual variables, which we talk about in §6 and §7.

5 Analysis I: A Weaker $O(K + \log n)$ Guarantee

We start with a simpler analysis which yields an $O(K+\log n)$-competitiveness. This argument will not use the increasing capacity assumption from Assumption 3; however, the result gives a competitiveness of $O(\max(K, \log n))$ which is logarithmic when K is small, whereas our eventual result will be $O(\min(K^{O(1)}, K + \log n))$, which can be much smaller when $K \ll \log n$.

▶ **Theorem 10.** *The scheduling algorithm in §3 is $O(K + \log n)$-competitive.*

Proof. For each job j, we arrange the tasks in $T(j)$ in descending order of their processing requirements. (This is the opposite of the order in which they finish, since all alive tasks of a job are processed at the same rate.) Say the sequence of the tasks for a job j is $v_1, \ldots, v_r$. We partition these tasks into *groups* with exponentially increasing cardinalities: $T_1(j) := \{v_1\}$, $T_2(j) := \{v_2, v_3\}$, and $T_h(j) := \{v_{2^{h-1}}, \ldots, v_{2^h-1}\}$ has 2^{h-1} tasks. (Assume w.l.o.g. that $r+1$ is a power of 2 by adding zero-sized tasks to $T(j)$). Now we define the dual variables.

Dual Variables. Define $\gamma := 2\max\{K, \log_2 n\}$.

- For a time t and machine i of speed class ℓ, let A^t denote the set of active (unfinished) jobs at time t, and define $\beta_{i,t} := \dfrac{w(A^t)}{m_\ell \cdot \gamma}$.
- For job j and a task $v \in T_h(j)$ in the h-th group, define $\delta_{j,v} := \dfrac{w_j}{2^{h-1} \cdot \gamma}$.
- In order to define $\alpha_{j,v}$, we first define quantities $\alpha_{j,v,t}$ for every time t, and then set $\alpha_{j,v} := \sum_t \alpha_{j,v,t}$. At time t, recall that $T^t(j)$ is the set of alive tasks of job j, and define

$$\alpha_{j,v,t} \;:=\; \frac{w_j}{|T^t(j)|} \cdot \mathbf{1}_{(v \text{ alive at time } t)} \;=\; \frac{w_j}{|T^t(j)|} \cdot \mathbf{1}_{(v \in T^t(j))} \;.$$

  This "spreads" the weight of j equally among its alive tasks.

Having defined the dual variables, we first argue that they are feasible.

▶ **Lemma 11** (Dual feasibility). *The dual variables defined above always satisfy the constraints* (9), (7) *and*(10) *for a speed-up factor $\gamma \geq 2\max\{K, \log_2 n\}$.*

Proof. To check feasibility of (7), consider a job j and observe that

$$\textstyle\sum_{v \in T(j)} \delta_{j,v} \;=\; \sum_h \sum_{v \in T_h(j)} \delta_{j,v} \;=\; \sum_h \sum_{v \in T_h(j)} \frac{w_j}{2^{h-1}\cdot\gamma} \;=\; \sum_h \frac{w_j}{\gamma} \;\leq\; w_j \;,$$

because $|T_h(j)| = 2^{h-1}$ and there are at most $\log_2 n \leq \gamma$ distinct groups. Feasibility of (9) also follows easily. It remains to check (10) for a job j, task v, machine i and times $t' \leq t$.

If v is not alive at time t', then $\alpha_{j,v,t'}$ is 0, and (10) follows trivially. Else, $v \in T^{t'}(j)$, and suppose $v \in T_h(j)$. This means the jobs in $T_1(j), \ldots, T_{h-1}(j)$ are also alive at time t', so $|T^{t'}(j)| \geq 1 + 2 + \ldots + 2^{h-2} + 1 = 2^{h-1}$. Furthermore, suppose the tasks in $T^{t'}(j)$ belong to block B (defined in §3.1), and let $\ell^\star$ be the speed class with the slowest machines among the associated machines $m(B)$. Let ℓ denote the speed class of machine i (considered in (10)). Two cases arise: the first is when $\ell \geq \ell^\star$, where $L_v^{t'} \geq \gamma\sigma_{\ell^\star} \geq \gamma\sigma_\ell = \gamma s_i$, so (10) holds because

$$\alpha_{j,v,t'} \;=\; \tfrac{w_j}{|T^{t'}(j)|} \;\leq\; \tfrac{w_j}{2^{h-1}} \;=\; \gamma \cdot \delta_{j,v} \;\leq\; \tfrac{\delta_{j,v} \cdot L_v^{t'}}{s_i} \;.$$

The second case is $\ell < \ell^\star$: Let $V \subseteq A^{t'}$ be the set of jobs which are frozen by the moment v freezes. In other words, V contains tasks in block B and the blocks before it. Applying the second statement in Corollary 8 with $V'' = V$,

$$\tfrac{w_j}{|T^{t'}(v)| L_v^{t'}} \;\leq\; \tfrac{\widetilde{w}^t(V)}{\sum_{v' \in V} L_{v'}^{t'}} \;\leq\; \tfrac{w(V)}{\sum_{v' \in V} L_{v'}^{t'}} \;\leq\; \tfrac{w(A^{t'})}{\gamma \cdot m_\ell \sigma_\ell} \;,$$

where the last inequality uses the fact that all machines of speed class ℓ are busy processing jobs in V. Therefore,

$$\alpha_{j,v,t'} \;=\; \tfrac{w_j}{|T^{t'}(j)|} \;\leq\; \tfrac{w(A^{t'}) \cdot L_v^{t'}}{\gamma \cdot m_\ell \sigma_\ell} \;\leq\; \tfrac{\beta_{i,t} \cdot L_v^{t'}}{s_i} \;,$$

the last inequality useing the definition of $\beta_{i,t}$ and that $w(A^t) \geq w(A^{t'})$. ◀

Finally, we show that the dual objective value for this setting of dual variables is close to the primal value. It is easy to check that $\sum_j \alpha_j = \sum_t w(A^t)$, which is the total weighted completion time of the jobs. Moreover,

$$\textstyle\sum_{i,t} \beta_{i,t} \;=\; \sum_t \sum_\ell \sum_{i : s_i = \sigma_\ell} \frac{w(A^t)}{m_\ell \, \gamma} \;=\; \frac{K}{\gamma} \cdot \sum_t w(A^t) \;.$$

Since we chose speedup $\gamma = 2\max\{K, \log_2 n\}$, we have $K \leq \gamma/2$ and the dual objective value $\sum_{j,v} \alpha_{j,v} - \sum_{i,t} \beta_{i,t}$ is at least half of the total weighted completion time (primal value). This completes the proof of Theorem 10. ◀

6 Analysis II: An Improved Guarantee for a Single Job

We want to show that the competitiveness of our algorithm just depends on K, the number of speed classes. To warm up, in this section we consider the special case of a single job; in §7 we consider the general case. As was shown in Proposition 5, any algorithm has competitive ratio $\Omega(K)$ even in the case of a single job. We give a matching upper bound using dual fitting for an instance *with a single job* j, say of weight 1, when the machines satisfy Assumption 3.

▶ **Theorem 12.** *If the machines satisfy Assumption 3, the scheduling algorithm in §3 is $O(K^2)$-competitive for a single job.*

6.1 The Intuition Behind the Improvement

The analysis in §5 incurred $\Omega(\log n)$-competitive ratio because we divided the execution of the tasks of each job into $O(\log n)$ *epochs*, where each epoch ended when the number of tasks halved. In each such epoch, we set the $\delta_{j,v}$ variables by distributing the job's weight evenly among all tasks alive at the beginning of the epoch. A different way to define epochs would

be to let them correspond to the time periods when the number of alive tasks falls in the range $(m_\ell, m_{\ell+1})$. This would give us only K epochs. There is a problem with this definition: as the number of tasks vary in the range $(m_\ell, m_{\ell+1})$, the rate assigned to tasks varies from σ_ℓ to $\sigma_{\ell+1}$. Indeed, there is a transition point $\widetilde{m}_\ell$ in $(m_\ell, m_{\ell+1})$ such that the rate assigned to the tasks stays close to $\sigma_{\ell+1}$ as long as the number of tasks lie in the range $(\widetilde{m}_\ell, \sigma_{\ell+1})$; but if the number of tasks lie in the range $(m_\ell, \widetilde{m}_\ell)$, the assigned rate may not stay close to any fixed value. However, in this range, the *total* processing rate assigned to all the tasks stays close to $m_\ell \sigma_\ell$.

It turns out that our argument for an epoch (with minor modifications) works as long as one of these two facts hold during an epoch: (i) the total rate assigned to the tasks stays close to $m_\ell \sigma_\ell$ for some speed class ℓ (even though the number of tasks is much larger than m_ℓ), or (ii) the actual rate assigned to the tasks stays close to σ_ℓ. Thus we can divide the execution of the job into $2K$ epochs, and get an $O(K)$-competitive algorithm. In this section, we prove this for a single job; we extend to the case of multiple jobs in §7 (with a slightly worse competitiveness).

6.2 Defining the New Epochs

Before defining the dual variables, we begin with a definition. For each speed class ℓ, define the *threshold* $\widetilde{m}_\ell$ to be the following:

$$\widetilde{m}_\ell := \tfrac{1}{\sigma_{\ell+1}} \left(\sigma_1 m_1 + \cdots + \sigma_\ell m_\ell\right). \tag{11}$$

The parameter $\widetilde{m}_\ell$ is such that the processing capacity of $\widetilde{m}_\ell$ machines of class $\ell+1$ equals the combined processing capacity of machines of class at most ℓ. The increasing capacity assumption implies $m_\ell < \widetilde{m}_\ell < m_{\ell+1}$, as formalized below:

▷ Claim 13. Define $M_\ell := m_1 + \ldots + m_\ell$ and $\widetilde{M}_\ell := M_\ell + \widetilde{m}_\ell$. Under the increasing capacity Assumption 3 and $\kappa = 2$, for any speed class ℓ, we have
(a) $2\widetilde{m}_\ell \le m_{\ell+1}$ and so, $\widetilde{M}_\ell \le M_{\ell+1}$,
(b) $\widetilde{M}_\ell \ge 2M_\ell$,
(c) $m_\ell \sigma_\ell \ge \frac{1}{2}\widetilde{m}_\ell \sigma_{\ell+1}$, and
(d) $\widetilde{m}_\ell \ge 2m_\ell$.

Proof. Fact (a) follows from the increasing capacity assumption and the definition of the threshold, since $2\widetilde{m}_\ell \sigma_{\ell+1} \le \sigma_{\ell+1} m_{\ell+1}$. This implies $\widetilde{M}_\ell = M_\ell + \widetilde{m}_\ell \le M_\ell + m_{\ell+1} \le M_{\ell+1}$. Proving (b) is equivalent to showing $\widetilde{m}_\ell \ge M_\ell$, which follows from the definition of $\widetilde{m}_\ell$ and the fact that $\sigma_{\ell+1} < \sigma_i$ for all $i \le \ell$. The last two statements also follow from the increasing capacity assumption. ◁

We identify a set of $2K$ *break-points* as follows: for each speed class ℓ, let t_ℓ denote the first time when M_ℓ alive tasks remain. Similarly, let $\widetilde{t}_\ell$ be the first time when exactly $\widetilde{M}_\ell$ alive tasks remain. Note that $t_{\ell+1} < \widetilde{t}_\ell < t_\ell$. Let $\widetilde{F}_\ell$ be the tasks which finish during $[t_{\ell+1}, \widetilde{t}_\ell]$, and F_ℓ be those which finish during $[\widetilde{t}_\ell, t_\ell]$. Let $\widetilde{f}_\ell$ and f_ℓ denote the cardinality of $\widetilde{F}_\ell$ and F_ℓ respectively. Note that $\widetilde{f}_\ell = M_{\ell+1} - \widetilde{M}_\ell = m_{\ell+1} - \widetilde{m}_\ell$, $f_\ell = \widetilde{M}_\ell - M_\ell = \widetilde{m}_\ell$.

▷ Claim 14. For any speed class ℓ, we have $f_\ell \le \widetilde{f}_\ell \le f_{\ell+1}$.

Proof. The first statement requires that $\widetilde{m}_\ell \le m_{\ell+1} - \widetilde{m}_\ell$. This is the same as $2\widetilde{m}_\ell \le m_{\ell+1}$, which follows from Claim 13 (a). The second statement requires that $m_{\ell+1} - \widetilde{m}_\ell \le \widetilde{m}_{\ell+1}$, i.e., $m_{\ell+1} \le \widetilde{m}_\ell + \widetilde{m}_{\ell+1}$. But $m_{\ell+1} \le \widetilde{m}_{\ell+1}$ (by Claim 13 (d)), hence the proof. ◁

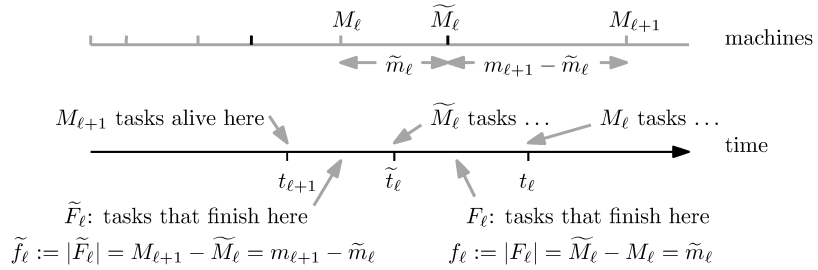

Figure 1 *Defining breakpoints.*

Next we set the duals. Although it is possible to directly argue that the delay incurred by the job in each epoch is at most (a constant times) the optimal objective value, the dual fitting proof generalizes to the arbitrary set of jobs.

6.3 Setting the Duals

Define the speed-up $\gamma \geq 2K$. We set the duals as:

- Define $\delta_{j,v} := \begin{cases} \frac{1}{2K \cdot f_\ell} & \text{if } v \in F_\ell \\ \frac{1}{2K \cdot \widetilde{f}_\ell} & \text{if } v \in \widetilde{F}_\ell \end{cases}$.
- For machine i of class ℓ, define $\beta_{i,t} := \frac{1}{2K \cdot m_\ell} \cdot \mathbf{1}_{(\text{not all tasks finished})}$.
- Finally, as in §5, we define $\alpha_{j,v,t}$ for each task v of job j, and then set $\alpha_{j,v} := \sum_t \alpha_{j,v,t}$. To define $\alpha_{j,v,t}$, we consider two cases (we use n_t to denote the number of alive tasks at time t):
  1. $n_t \in [M_\ell, \widetilde{M}_\ell)$ for some ℓ: Then $\alpha_{j,v,t} := (1/n_t) \cdot \mathbf{1}_{(v \text{ alive at time } t)}$.
  2. $n_t \in [\widetilde{M}_\ell, M_{\ell+1})$ for some ℓ: Then $\alpha_{j,v,t} := (1/f_\ell) \cdot \mathbf{1}_{(v \in F_\ell)}$.

  Note the asymmetry in the definition. It arises because in the first case, the *total* speed of machines processing a task is (up to a constant) $m_\ell \sigma_\ell$, whereas in the second case the *average* speed of such machines is about $\sigma_{\ell+1}$.

▶ **Lemma 15** (Dual feasibility). *The dual variables defined above always satisfy the constraints (7) and* (9)*, and satisfy constraint (10) for speed-up $\gamma \geq 2K$.*

Proof. It is easy to check from the definition of $\delta_{j,v}$ and $\alpha_{j,v,t}$ that the dual constraints (7) and (9) are satisfied. It remains to verify constraint (10) (re-written below) for any task v, machine i, times t and $t' \geq t$.

$$\alpha_{j,v,t'} \leq \frac{L_v^{t'}}{s_i} \cdot (\beta_{i,t} + \delta_{j,v}) \ . \qquad \text{((10) repeated)}$$

As in the definition of $\alpha_{j,v,t'}$, there are two cases depending on where $n_{t'}$ lies. First assume that there is class $\ell^\star$ such that $M_{\ell^\star} \leq n_{t'} < \widetilde{M}_{\ell^\star}$. Assume that v is alive at time t' (otherwise $\alpha_{j,v,t'}$ is 0), so $\alpha_{j,v,t'} = \frac{1}{n_{t'}}$, where $n_{t'}$ is the number of alive tasks at time t'. Being alive at this time t', we know that v will eventually belong to some F_ℓ with $\ell \leq \ell^\star$, or in some $\widetilde{F}_\ell$ with $\ell < \ell^\star$. So by Claim 14, $\delta_{j,v} \geq \frac{1}{2K \cdot f_{\ell^\star}}$. Moreover, let i be a machine of some class ℓ, so $s_i = \sigma_\ell$. Hence, it is enough to verify the following in order to satisfy (10):

$$\frac{1}{n_{t'}} \quad \leq \quad \frac{L_v^{t'}}{\sigma_\ell} \cdot \left(\frac{1}{2K \cdot m_\ell} + \frac{1}{2K \cdot f_{\ell^\star}} \right) . \qquad (12)$$

Two subcases arise, depending on how ℓ and $\ell^\star$ relate – in each we show that just one of the terms on the right is larger than the left.

- $\ell^\star \geq \ell$: Since at least $M_{\ell^\star}$ tasks are alive at this time, the total speed assigned to all the alive tasks at time t' is at least $\gamma \cdot \sigma_{\ell^\star} m_{\ell^\star}$. Therefore, $L_v^{t'} \geq \frac{\gamma \cdot m_{\ell^\star} \sigma_{\ell^\star}}{n_{t'}}$. Now using $\gamma \geq 2K$, we get

$$\frac{L_v^{t'}}{2K \cdot m_\ell \sigma_\ell} \quad \geq \quad \frac{m_{\ell^\star}\sigma_{\ell^\star}}{m_\ell \sigma_\ell} \cdot \frac{1}{n_{t'}} \quad \geq \quad \frac{1}{n_{t'}} \ ,$$

  where the last inequality follows from the increasing capacity assumption.
- $\ell^\star \leq \ell - 1$: The quantity $L_v^{t'} n_{t'}$ is the total speed of the machines which are busy at time t', which is at least $\gamma(m_1\sigma_1 + \ldots + m_{\ell^\star}\sigma_{\ell^\star}) = \gamma \cdot \widetilde{m}_{\ell^\star}\sigma_{\ell^\star+1}$. Again, using $\gamma \geq 2K$, we get

$$\frac{L_v^{t'} \cdot n_{t'}}{2K \cdot f_{\ell^\star}\sigma_\ell} \quad \geq \quad \frac{\widetilde{m}_{\ell^\star}\sigma_{\ell^\star+1}}{f_{\ell^\star}\sigma_\ell} \quad \geq \quad 1$$

  because $\sigma_{\ell^\star+1} \geq \sigma_\ell$ and $\widetilde{m}_{\ell^\star} = f_{\ell^\star}$.

Thus, (12) is satisfied in both the above subcases.

Next we consider the case when there is a speed class $\ell^\star$ such that $\widetilde{M}_{\ell^\star} < n_{t'} \leq M_{\ell^\star+1}$. We can assume that $v \in F_{\ell^\star}$, otherwise $\alpha_{j,v,t'}$ is 0; this means $\delta_{v,j} = \frac{1}{2K \cdot f_{\ell^\star}}$. Since $\alpha_{j,v,t'} = \frac{1}{f_\ell} = \frac{1}{\widetilde{m}_{\ell^\star}}$, and $L_v^{t'} \geq \gamma \cdot \sigma_{\ell^\star+1}$, the expression (10) follows from showing

$$\frac{1}{\widetilde{m}_{\ell^\star}} \quad \leq \quad \frac{\gamma}{\sigma_\ell} \cdot \left(\frac{1}{2K \cdot m_\ell} + \frac{1}{2K \cdot f_{\ell^\star}} \right) \cdot \sigma_{\ell^\star+1} \ . \tag{13}$$

Since $\gamma \geq 2K$, we can drop those terms. Again, two cases arise:

- $\ell^\star \geq \ell$: By definition, $\sigma_{\ell^\star+1} \cdot \widetilde{m}_{\ell^\star} \geq \sigma_{\ell^\star} m_{\ell^\star} \geq \sigma_\ell m_\ell$ (by the increasing capacity assumption).
- $\ell^\star \leq \ell - 1$: Since $f_{\ell^\star} = \widetilde{m}_{\ell^\star}$ and $\sigma_\ell \leq \sigma_{\ell^\star+1}$, this case also follows easily. ◀

Proof of Theorem 12. Having checked dual feasibility in Lemma 15, consider now the objective function. For any time t when at least one task is alive, $\sum_v \alpha_{j,v,t} = 1$. Therefore, $\sum_v \alpha_{j,v}$ is the makespan. Also, $\sum_i \beta_{i,t} = 1/2$ as long as there are unfinished tasks, so $\sum_{i,t} \beta_{i,t}$ is half the makespan, and the objective function $\sum_v \alpha_{j,v} - \sum_{i,t} \beta_{it}$ also equals half the makespan. Since we had assumed $\gamma = O(K)$-speedup, the algorithm is $O(K)$-competitive. ◀

7 Analysis III: Proof for $\tilde{O}(K^3)$ Guarantee

We now extend the ideas from the single job case to the general case. We only discuss the proof outline here, and refer the readers to the full version for details. For time t, let A^t be the set of alive jobs at time t. Unlike the single job case where we had only one block, we can now have multiple blocks. While defining $\alpha_{j,v,t}$ in the single job case, we had considered two cases: (i) the rate assigned to each task stayed close to σ_ℓ for some class ℓ (this corresponded to $n_t \in [\widetilde{M}_{\ell-1}, M_\ell)$), and (ii) the total rate assigned to each task was close to $m_\ell\sigma_\ell$ for speed class ℓ (this corresponded to $n_t \in [M_\ell, \widetilde{M}_\ell)$). We extend these notions to blocks as follows:

Simple blocks: A block B is said to be *simple* w.r.t. to a speed class ℓ if the average rate assigned to the tasks in B is close to σ_ℓ. Similarly a job j is said to be simple w.r.t. a speed class ℓ if all the alive tasks in it are assigned rates close to σ_ℓ (recall that all alive tasks in a job are processed at the same rate). All the jobs in a simple block B may not be simple (w.r.t. the same speed class ℓ), but we show that a large fraction of jobs (in terms of weight) in B will be simple. Thus, it is enough to account for the weight of simple jobs in B. This is analogous to case (i) mentioned above (when there is only one job and tasks in it receive rate

close to σ_ℓ). In §6, we had defined $\alpha_{j,v,t}$ for such time t as follows: we consider only those tasks which survive in F_ℓ, and then evenly distribute w_j among these tasks. The analogous definition here would be as follows: let $\tau_{\ell,j}$ be the *last* time when j is simple w.r.t. the speed class ℓ. We define $\alpha_{j,v,t}$ by evenly distributing w_j among those tasks in v which are alive at $\tau_{\ell,j}$. We give details in the full version.

Long blocks: The total speed of the machines in this block stays close to $m_\ell\sigma_\ell$ for some speed class ℓ. Again, inspired by the definitions in §6, we assign $\alpha_{j,v,t}$ for tasks $v \in B$ by distributing $w(B)$ to these tasks (in proportion to the rate assigned to them). From the perspective of a job j which belongs to a long block B w.r.t. a speed class σ_ℓ at a time t, the feasibility of (6) works out provided for *all* subsequent times t' when j again belongs to such a block B', we have $w(B')$ and $w(B)$ remain close to each other. If $w(B')$ exceeds (say) $2w(B)$, we need to reassign a new set of $\delta_{j,v}$ values for v. To get around this problem we require that long blocks (at a time t) also have weight at least $w(A^t)/(10K)$. With this requirement, the doubling step mentioned above can only happen $O(\log K)$ times (and so we incur an additional $O(\log K)$ in the competitive ratio). The details are given in the full version. Blocks which were cheaper than $w(A^t)/(10K)$ do not create any issue because there can be at most K of them, and so their total weight is small in comparison to $w(A^t)$.

Short blocks: Such blocks B straddle two speed classes, say ℓ and $\ell+1$, but do not contain too many machines of either class (otherwise they will fall into one of the two categories above). We show in the full version that the total weight of such blocks is small compared to $w(A^t)$. The intuitive reason is as follows: for any two consecutive short blocks B_1 and B_2, there must be blocks in between them whose span is much longer than B_2. Since these blocks freeze before B_2, their total weight would be large compared to $w(B_2)$.

In the overall analysis, we charge short blocks to simple and long blocks, and use dual fitting as indicated above to handle simple and long blocks.

8 Discussion

Several interesting problems remain open. (i) Can we close the gap between lower bound of $\Omega(K)$ and upper bound of $O(K^3 \log^2 K)$? (ii) Can we prove an analogous result for weighted *flow-time* (with speed augmentation)? (iii) Can we generalize this result to the unrelated machines setting? (iv) Our lower bound of $\Omega(K)$-competitive ratio relies on non-clairvoyance; can we prove a better bound if the processing times of tasks are known at their arrival times?

References

1. Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallel DAG jobs online to minimize average flow time. In *Proceedings of SODA*, pages 176–189, 2016.
2. C. Anglano and M. Canonico. Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, 2008.
3. Abbas Bazzi and Ashkan Norouzi-Fard. Towards tight lower bounds for scheduling problems. In *Proceedings of ESA*, pages 118–129, 2015.
4. Anne Benoit, Loris Marchal, Jean-Francois Pineau, Yves Robert, and Frédéric Vivien. Scheduling concurrent bag-of-tasks applications on heterogeneous platforms. *IEEE Trans. Computers*, 59(2):202–217, 2010.
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 José R. Correa, Martin Skutella, and José Verschae. The power of preemption on unrelated machines and applications to scheduling orders. In *Proceedings of APPROX/RANDOM*, pages 84–97, 2009.

7 Naveen Garg, Anupam Gupta, Amit Kumar, and Sahil Singla. Non-clairvoyant precedence constrained scheduling. In *Proceedings of ICALP*, pages 63:1–63:14, 2019.

8 Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22(3):513–544, 1997.

9 Shi Li. Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In *Proceedings of FOCS*, pages 283–294. 2017.

10 Ioannis A. Moschakis and Helen D. Karatza. Multi-criteria scheduling of bag-of-tasks applications on heterogeneous interlinked clouds with simulated annealing. *J. Syst. Softw.*, 101:1–14, 2015.

11 Alix Munier, Maurice Queyranne, and Andreas S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In *Proceedings of IPCO*, volume 1412, pages 367–382. 1998.

12 Maurice Queyranne and Maxim Sviridenko. A (2+epsilon)-approximation algorithm for generalized preemptive open shop problem with minsum objective. In *Proceedings of IPCO*, volume 2081, pages 361–369, 2001.

13 Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *Proceedings of SODA*, pages 491–500, 2008.

A Missing Proofs of Section 2

▶ **Proposition 4.** *An arbitrary instance can be transformed into one satisfying Assumption 3 by losing a factor $O(\gamma K)$ in the competitive ratio.*

Proof. We show how to transform the instance so that it satisfies the increasing capacity assumption, while losing only $O(K)$-factor in the competitive ratio. For sake of brevity, let κ denote the constant 64.

For a speed class ℓ, let C_ℓ denote $m_\ell \sigma_\ell$, i.e., the total processing capacity of the machines in this speed class. Starting from speed class 1, we construct a subset X of speed classes as follows: if ℓ denotes the last speed class added to X, then let $\ell' > \ell$ be the smallest class such that $C_{\ell'} \geq 2\kappa C_\ell$. We add ℓ' to X and continue this process till we have exhausted all the speed classes.

Consider the instance $\mathcal{I}'$ in which the set of jobs is the same as those in $\mathcal{I}$, but there are Km_ℓ machines of speed class ℓ for each $\ell \in X$. For a speed class $\ell \in X$, let C'_ℓ denote $2\kappa K m_\ell \sigma_\ell$, which is at most the total capacity of the speed class ℓ machines in $\mathcal{I}'$. Let us now consider the optimal solutions of the two instances. We first observe that $\mathsf{opt}(\mathcal{I}') \leq \mathsf{opt}(\mathcal{I})$. Consider two consecutive speed classes $\ell_1 < \ell_2$ in X. From the definition of X, we see that $C'_{\ell_1} \geq \sum_{l=\ell_1}^{\ell_2 - 1} C_l$. Therefore all the processing done by a solution to $\mathcal{I}$ on machines of speed class $[\ell_1, \ell_2)$ during a timeslot $[t, t+1]$ can be performed on machines of speed class ℓ_1 in $\mathcal{I}'$ during the same timeslot. Therefore, $\mathsf{opt}(\mathcal{I}') \leq \mathsf{opt}(\mathcal{I})$.

For the converse statement, it is easy to see that if we give $2\kappa K$ speedup to each machine in $\mathcal{I}$, then the processing capacity of each speed class in $\mathcal{I}$ is at least that in $\mathcal{I}'$. Therefore, $\mathsf{opt}(\mathcal{I}) \leq 2\kappa K \mathsf{opt}(\mathcal{I}')$. Therefore, replacing $\mathcal{I}$ by $\mathcal{I}'$ will result in $O(\kappa K)$ loss in competitive ratio. It is also easy to check that $\mathcal{I}'$ satisfies increasing capacity assumption.

Observe that the conversion from $\mathcal{I}$ to $\mathcal{I}'$ can be easily done at the beginning – we just need to identify the index set X, and use only these for processing. The factor K loss in competitive ratio is also tight for the instance $\mathcal{I}$ where all speed classes have the same capacity. ◀

B Missing proofs of Section 3

▶ **Lemma 6.** *A schedule $\mathcal{S}$ is feasible if for every time t and every value of k:*

($\star$) *the total rate assigned to any subset of k tasks is at most $\gamma \cdot S_k$.*

Proof. The rates assigned to tasks change only when one of these events happen: (i) a new job j arrives, (ii) an existing task finishes. Assuming that the job sizes, release dates are integers, we can find a suitable $\delta > 0$ (which will also depend on the speeds of the machines) such that all the above events happen at integral multiples of δ.

Consider an interval $[t, t+\delta)$, where t is an integral multiple of δ. We need to show that if L_v^t's satisfy the condition ($\star$), then we can build a feasible schedule during $[t, t+\delta)$. By feasibility, we mean that each task v can be processed to an extent of $\bar{p}_v := L_v^t \cdot \delta$ extent and at any point of time, it gets processed on at most one machine.

We follow a greedy strategy to build the schedule. Suppose we have built the schedule till time $t' \in [t, t+\delta)$. At time t', we order the tasks in descending order of the remaining processing requirement for this slot (at time t, each task v has processing requirement of $\bar{p}_v$). Let the ordered tasks at time t' be $v_1, \ldots, v_n$. We schedule v_i on machine i.

Suppose for the sake of contradiction, a task $v^\star$ is not able to complete $\bar{p}_{v^\star}$ amount of processing. We first make the following observation:

▷ Claim 16. Let v and v' be two tasks such that at some time $t' \in [t, t+\delta)$, we prefer v to v' in the ordering at time t'. Then if v' does not complete $\bar{p}_{v'}$ amount of processing during $[t, t+\delta)$, then neither does v.

Proof. Since we prefer v at time t', v has more remaining processing time. If we never prefer v' to v after time t', then v always has more remaining processing requirement than v' during this interval. If we prefer v' to v at some point of time during $(t', t+\delta)$, then it is easy to check that the remaining processing requirements for both v and v' will remain the same. The result follows easily from this observation. ◁

Starting from $\{v^\star\}$, we build a set S of tasks which has the following property: if $v \in S$, then we add to S all the tasks v' such that v' was preferred over v at some point of time during $[t, t+\delta)$. Repeating application of Claim 16 shows that none of these tasks v complete $\bar{p}_v$ amount of processing during $[t, t+\delta)$. Let $\bar{m}$ denote $|S|$. We note that only tasks in S would have been processed on the first $\bar{m}$ machines during $[t, t+\delta)$ – otherwise, we can add more tasks to S. Since none of these tasks finish their desired amount of processing during this interval, it follows that

$$\sum_{v \in S} \bar{p}_v \geq \gamma\delta \cdot S_{\bar{m}} \ .$$

Since $\bar{p}_v = \delta L_v^t$, we see that the set of tasks in S violates ($\star$). This is a contradiction, and so such a task $v^\star$ cannot exist. ◀

▶ **Corollary 8.** *Fix a time t. Let V be the set of tasks frozen by some moment τ. For a task $v \in V$,*

(i) *if $V' \subseteq V$ is any subset of tasks which freeze either at the same moment as v, or after it, then $\frac{\widetilde{w}^t(v)}{s_{|V|}} \geq \frac{w(V')}{S_{|V|}}$.*

(ii) *if $V'' \subseteq V$ is any subset of tasks which freeze either at the same moment as v, or before it, then $\frac{\widetilde{w}^t(v)}{L_v^t} \leq \frac{\widetilde{w}^t(V'')}{\sum_{v' \in V''} L_{v'}^t}$.*

Proof. For any task v', let $\tau_{v'}$ be the value of τ at which v' freezes. We know that

$$\sum_{v' \in V} \widetilde{w}^t(v') \cdot \tau_{v'} = \gamma S_{|V|} \ . \tag{14}$$

Since $\sum_{v' \in V \setminus \{v\}} \widetilde{w}^t(v') \cdot \tau_{v'} \leq \gamma S_{|V|-1}$ by feasibility, it follows that

$$\widetilde{w}^t(v) \cdot \tau_v \geq \gamma s_{|V|} \ . \tag{15}$$

Now for all $v' \in V'$, we have $\tau_{v'} \geq \tau_v$, so

$$\widetilde{w}^t(V') \cdot \tau_v \ = \sum_{v' \in V'} \widetilde{w}^t(v') \cdot \tau_v \ \leq \sum_{v' \in V'} \widetilde{w}^t(v') \cdot \tau_{v'} \ \leq \sum_{v' \in V} \widetilde{w}^t(v') \cdot \tau_{v'} \ \overset{(14)}{=} \ \gamma S_{|V|} \ .$$

Hence, the first claim follows:

$$\frac{\widetilde{w}^t(v)}{s_{|V|}} \ \overset{(15)}{\geq} \ \frac{\gamma}{\tau_v} \ \geq \ \frac{\widetilde{w}^t(V')}{S_{|V|}} \ .$$

For the second claim,

$$\sum_{v' \in V''} L^t_{v'} \ = \ \sum_{v' \in V''} \widetilde{w}^t_{v'} \cdot \tau_{v'} \ \leq \ \widetilde{w}^t(V'') \cdot \tau_v \ .$$

The claim now follows by the definition $L^t_v = \widetilde{w}^t(v) \cdot \tau_v$. ◀

C Missing Proofs of Section 4

▷ Claim 9. For any schedule $\mathcal{S}$, there is a feasible solution to the LP of objective value at most $2\,\mathsf{cost}(\mathcal{S})$.

Proof. Consider a schedule $\mathcal{S}$, and let x_{ivt} be the extent of processing done on a task v (belonging to job j) during $[t, t+1]$ on machine i. More formally, if the task is processed for ε units of time on machine i during this time slot, then we set x_{ivt} to $\varepsilon \cdot s_i$. Constraint (4) states that every task v needs to be processed to an extent of p_v, whereas (5) requires that we cannot do more than s_i unit of processing in a unit time slot on machine i. Now we verify verify (3). Consider a task job j and a task v belonging to it. The total processing time of v is

$$\sum_{i,t} \frac{x_{ivt}}{s_i}. \tag{16}$$

The completion time F_j of j is at least the processing time of each of the tasks in it. Finally, we check (2). Define $F_{j,t}$ to be 1 if j is alive at time t. The RHS of this constraint is the fraction of v which is done after time t; and so if this is non-zero, then $F_{j,t}$ is 1. This shows the validity of this constraint.

In the objective function, the first term is the total weighted completion time of all the jobs. The second term is also the same quantity, because F_j is equal to $\sum_{t \geq r_j} F_{j,t}$. ◁