Non-Preemptive Speed Scaling

Antonios Antoniadis[*] Chien-Chung Huang[*]

Abstract

We consider the following variant of the speed scaling problem introduced by Yao, Demers, and Shenker. We are given a set of jobs and we have a variable-speed processor to process them. The higher the processor speed, the higher the energy consumption. Each job is associated with its own release time, deadline, and processing volume. The objective is to find a feasible schedule that minimizes the energy consumption. Moreover, no preemption of jobs is allowed.

Unlike the preemptive version that is known to be in P, the non-preemptive version of speed scaling is strongly NP-hard. In this work, we present a constant factor approximation algorithm for it. The main technical idea is to transform the problem into the unrelated machine scheduling problem with L_p-norm objective.

1 Introduction

The *speed scaling* problem was introduced by Yao, Demers and Shenker [16] in 1995. The input is a set $\mathcal{J}$ of jobs. Each job $j \in \mathcal{J}$ is associated with a release time r_j, a deadline d_j and a volume v_j. W.l.o.g., we assume that release times and deadlines are given as integers. We have a variable-speed processor, which is associated with a *power function* $P(s) = s^\alpha$ with $\alpha > 1$. Assuming that job $j \in \mathcal{J}$ is always processed at a speed of s_j, then j requires v_j/s_j time to be completed. The energy consumption of the processor is power integrated over time. A schedule is said to be feasible if the volume of each job is completely processed not before its release time but before its deadline. The goal is to find a feasible schedule that minimizes the total energy consumption.

Given our growing awareness of the environment, speed scaling is a natural problem to study. Practically, many modern microprocessors, in order to be more energy-efficient, have the built-in capability to vary their speed. It is desirable to have good scheduling policies in order to reduce the energy consumption in such computing environments. Furthermore this can help prolonging battery lives in mobile devices.

The original model of Yao et al. assumes that jobs can be preempted. That is, the execution of a job may be paused and resumed at a later point in time. The *non-preemptive* version of this problem, where a job, once started, must be processed until its completion, has not been studied so far. We call the problem *non-preemptive speed scaling.* The assumption that preemption is disallowed is a natural one to make, since preemption is known to cause significant overhead in practice. The fact that some theoretically good scheduling algorithms perform relatively bad in practice has been attributed to this overhead [10, 11, 13]. Furthermore, since the preemption of a job may require storing the state of both the system and the processing of the job, the implementation of preemption can become particularly hard in some settings. Some researchers try to cope with the above problems by either limiting or completely avoiding preemption (see [4, 5] for some examples).

Unlike the preemptive speed scaling, for which Yao et al. gave a simple and elegant polynomial time algorithm, the non-preemptive speed scaling problem is NP-hard. To see this, consider the following reduction from the *partition* problem [12]. In the partition problem, a set of integers $N = \{n_1, n_2, \cdots\}$ is given. The question is whether the set N can be partitioned into two disjoint subsets, so that the sum of one subset is equal to the sum of the other subset. We create a job j with release time 1 and deadline 2. Its volume v_j is $\sum_i n_i$. Further, for each integer $n_t \in N$, create a job of volume v_t with release time 0 and deadline 3. Due to the fact that the power function $P(s) = s^\alpha$ is strictly convex, it is easy to see that the original instance allows a partition if and only if the optimal schedule uses the same speed in the intervals $[0, 1)$ and $[2, 3)$. By a straightforward generalization of the idea, we can reduce from the *3-Partition* problem [12] to show that non-preemptive speed scaling is strongly NP-hard.

[*]Department of Computer Science, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin. {antoniad, villars}@informatik.hu-berlin.de

We note that even though the general case of the problem is NP-hard, there is a natural special case that is in P: when the given instance has *agreeable deadlines*. An instance has agreeable deadlines if given any two jobs $j, j' \in \mathcal{J}$ with $r_j < r_{j'}$, then $d_j \leq d_{j'}$. In words, earlier-released jobs also have earlier deadlines. For this special case, the original algorithm of Yao et al. can be applied by the observation that the schedule returned does not make use of preemption. However, to deal with the general case, novel algorithmic ideas are required.

Our Contribution and Technique

We develop a constant-factor approximation algorithm for the problem of non-preemptive speed scaling.

Theorem 1. *There exists a polynomial time algorithm that achieves a $2^{5\alpha-4}$-approximation for the non-preemptive speed scaling problem. For the special case of* laminar instances*, there exists a polynomial time algorithm that achieves a $2^{4\alpha-3}$-approximation.*

An instance is said to be a laminar instance if for any two jobs $j, j' \in \mathcal{J}$, either $[r_j, d_j) \subseteq [r_{j'}, d_{j'})$, $[r_{j'}, d_{j'}) \subseteq [r_j, d_j)$, or $[r_j, d_j) \cap [r_{j'}, d_{j'}) = \emptyset$. Laminar instances are of interest for two reasons. First, they form a natural special class of instances. As reported in [14], when the jobs are created by recursive calls in a program, the resulting instance is laminar. Secondly, and more importantly, we observe that a laminar instance is in a sense the "opposite" of an instance with agreeable deadlines: for any two jobs that have overlapping intervals, the one with earlier release time must have a latter deadline. As the agreeable version can be solved in polynomial time, this "opposite" version highlights the difficulty of the non-preemptive speed scaling problem. (Observe that the reductions we used earlier from partition/3-partition are laminar instances as well).

In Section 3, we show how to achieve a $2^{4\alpha-3}$ approximation for laminar instances. The rough idea is to use a series of transformations, which involve the "chopping up" of the intervals of the jobs, so that we can reduce a laminar instance into an instance of unrelated machine scheduling with the L_α-norm objective, which is known to be 2-approximable [2]. The "chopping up" of an interval, roughly speaking, means that we create new instances with the additional constraint that a job has to be processed entirely within a sub-interval. The technical challenge is to bound the growth of the cost throughout the series of transformations.

In Section 4, we develop a sweepline algorithm and use a certain "energy-folding" technique, to reduce a general instance into a laminar instance, at a further cost of $2^{\alpha-1}$ in the approximation factor.

Due to space constraints some of the proofs are omitted.

Previous work

Even though dynamic speed scaling has been studied extensively over the past years, to the best of our knowledge, non-preemptive speed scaling was not considered before. As already mentioned, the study of the (preemptive) problem was initiated by Yao et al. [16], who have developed a polynomial-time algorithm for the offline problem. Some special cases of the preemptive version of the problem have also been studied. For example, when the given instance is laminar [14] or when the power function is discrete [15].

In [8] and [9] constant-factor approximation algorithms for the problem on fixed-speed processors were given. In [8], more than one processor may be used in order to feasibly schedule the jobs, and the objective is to minimize the number of used processors. In [9] one may skip jobs in order to produce a feasible schedule. The objective here is to maximize the number of scheduled jobs. In [7] a similar problem of scheduling jobs without preemption on a variable-speed processor was studied. However in their model the processor can only operate at discrete speed levels, and there are precedence constraints among the jobs. They show that their problem is NP-hard and give polynomial time approximation schemes for two restricted cases.

So far we only discussed the offline version of the speed scaling problem. The online version has also been extensively studied (but still only for the preemptive version). Yao et al. [16] presented two online algorithms called *Average Rate* and *Optimal Available*. For the first they have proven a competitive ratio of $(2\alpha)^\alpha/2$ whereas the second was analyzed by Bansal, Kimbrel and Pruhs [3] who have shown a tight competitive ratio of α^α.

For a recent literature review on energy efficient algorithms, see [1].

2 Preliminaries

We define a more general version of our problem. Let $\mathcal{J}$ be the set of jobs. Each job $j \in \mathcal{J}$ has a set of disjoint *allowed intervals*, $I_{j1} = [r_{j1}, d_{j1})$, $I_{j2} = [r_{j2}, d_{j2})$, $\cdots$. A schedule $\mathcal{S}$ is *feasible* if each job $j \in \mathcal{J}$ is executed *entirely within* one of its allowed intervals. Note that in the original problem instance $\mathcal{I}$, each job has only one allowed interval. We will transform $\mathcal{I}$ so that a job may have multiple intervals. Let $d_{\max}$ be the latest deadline in the original instance $\mathcal{I}$. Assume without loss of generality, that the earliest release time among all the jobs in $\mathcal{J}$ is 0.

Throughout the article, we implicitly assume that a feasible schedule processes a job using a uniform speed. This assumption is without loss of generality, since the power function $P(s) = s^\alpha$ is strictly convex and using a uniform speed for a given job minimizes its expended energy.

Given a schedule $\mathcal{S}$, let $\mathcal{S}(j) = [b_j, e_j)$ denote the execution interval of job j and $e_j(\mathcal{S})$ the finishing time of job j under schedule $\mathcal{S}$. Furthermore, let $E(\mathcal{S})$ denote the total energy spent by schedule $\mathcal{S}$ and $E(\mathcal{S}, j)$ be the energy used for processing job j under schedule $\mathcal{S}$. The following proposition follows from our assumption that a feasible schedule processes each job at uniform speed.

Proposition 1. *Suppose that schedules $\mathcal{S}$ and $\mathcal{S}'$ process job j with speed s and s' respectively. Ifs $\leq cs'$ for some $c \geq 1$, then $E(\mathcal{S}, j) \leq c^{\alpha-1} E(\mathcal{S}', j)$.*

Proof. Recall that v_j is the volume of job j. Then

$$E(\mathcal{S}, j) = P(s)\frac{v_j}{s} = v_j s^{\alpha-1} \leq v_j c^{\alpha-1}\frac{(s')^\alpha}{s'} = c^{\alpha-1} P(s')\frac{v_j}{s'} = c^{\alpha-1} E(\mathcal{S}', j).$$

□

3 Special Case: Laminar Family

In this section we assume that the given instance $\mathcal{I}$ is a laminar instance. Recall that for such an instance, for any two jobs $j, j' \in \mathcal{J}$, either $[r_j, d_j) \subseteq [r_{j'}, d_{j'})$, $[r_{j'}, d_{j'}) \subseteq [r_j, d_j)$, or $[r_j, d_j) \cap [r_{j'}, d_{j'}) = \emptyset$. We can associate the jobs in $\mathcal{I}$ with a tree structure as follows. A job j is a descendant of another job j' in the tree if and only if $[r_j, d_j) \subset [r_{j'}, d_{j'})$. A job j is said to be a *leaf job* if its interval $[r_j, d_j)$ is not a proper superset of the interval of any other job. In case that there are more than one candidate leaf jobs with identical intervals $[r_j, d_j)$, we pick an arbitrary one of them as a leaf job. Let the leaf jobs form a set L. Note that all the jobs in L have distinct deadlines.

Transformation I

Our first step is to partition the entire interval $[0, d_{\max})$ into "zones" using a set of "landmarks."

Definition 1. *Let $\tau_1 < \tau_2 < \cdots < \tau_{|L|}$ be the set of ordered deadlines d_j for the jobs $j \in L$. Furthermore, let $\tau_0 = 0$ and $\tau_{|L|+1} = d_{\max}$. (Note that $\tau_0 < \tau_1$ and $\tau_{|L|} \leq \tau_{|L|+1}$.) The intervals $[\tau_i, \tau_{i+1})$ for all $0 \leq i \leq |L|$ are called* zones. *Throughout the text, we will refer to the τ_i's as* landmarks.

We create a new instance $\mathcal{I}_1$ as follows. Consider a job $j \in \mathcal{J}$ with its allowed interval $I_j = [r_j, d_j)$, such that

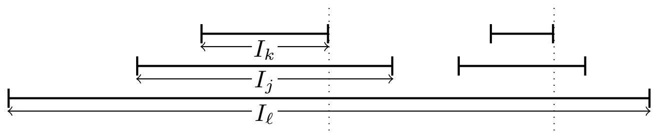
$$\tau_{i-1} \leq r_j < \tau_i < \tau_{i+1} < \cdots < \tau_{i+k} < d_j \leq \tau_{i+k+1}.$$

We replace its allowed interval $I_j = [r_j, d_j)$ with a set of allowed intervals $\bigcup_{s=1}^{k+2} I_{js}$, where

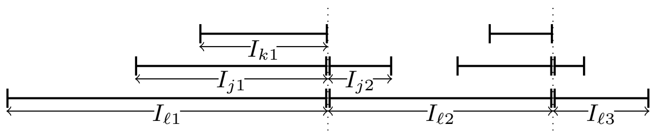
$$I_{j1} = [r_j, \tau_i),\ I_{j(k+2)} = [\tau_{i+k}, d_j),\ \text{and } I_{js} = [\tau_{s+i-2}, \tau_{s+i-1}) \text{ for } 2 \leq s \leq k+1.$$

See Figure 1 for an illustration. We now show that the above partition of jobs' allowed intervals does not increase the cost of an optimal solution by too much.

Lemma 1. *Let $OPT_{\mathcal{I}}$ and $OPT_{\mathcal{I}_1}$ denote the optimal schedules for instances $\mathcal{I}$ and $\mathcal{I}_1$ respectively. Then $E(OPT_{\mathcal{I}_1}) \leq 2^{\alpha-1} E(OPT_{\mathcal{I}})$.*



(a) The original instance $\mathcal{I}$.



(b) The modified instance $\mathcal{I}_1$.

Figure 1: In (a), the two dotted lines are the landmarks, since they are the deadlines of the leaf jobs. In (b), we show how to use the landmarks to divide a job's allowed interval in $\mathcal{I}$ into several allowed intervals in the modified instance $\mathcal{I}_1$.

Proof. We prove the lemma by transforming $OPT_{\mathcal{I}}$ into a feasible schedule $\overline{OPT}_{\mathcal{I}_1}$ for instance $\mathcal{I}_1$ and argue that $E(\overline{OPT}_{\mathcal{I}_1}) \leq 2^{\alpha-1}E(OPT_{\mathcal{I}})$.

First observe that for any job j, its execution in $OPT_{\mathcal{I}}$ cannot strictly contain a zone, unless it is the last zone. That is,

$$OPT_{\mathcal{I}}(j) \not\supset [\tau_{i-1}, \tau_i), 1 \leq i \leq |L|.$$

This holds because every zone $[\tau_{i-1}, \tau_i)$ contains a leaf job's entire interval $I_{j'}$ for $1 \leq i \leq |L|$. (Notice that the last zone does not necessarily contain the interval of a leaf job.) If $OPT_{\mathcal{I}}(j)$ contains such a zone, then job j' is not processed in $OPT_{\mathcal{I}}$, a contradiction.

By this observation, $OPT_{\mathcal{I}}(j) = [b_j, e_j)$ crosses at most one landmark τ_i. In our transformation we do not change the execution intervals for any job j that does not cross any landmark τ_i, i.e., $\overline{OPT}_{\mathcal{I}_1}(j) = OPT_{\mathcal{I}}(j)$. Observe that these jobs are completely processed within one of their allowed intervals in $\mathcal{I}_1$ and their processing speeds remain the same.

Next suppose that a job j crosses a landmark $\tau_i \in [b_j, e_j)$. Without loss of generality, we can assume that

$$\tau_i - b_j \geq e_j - \tau_i, \tag{1}$$

in other words, job j does not get processed for more time after τ_i than it does before τ_i.

$\overline{OPT}_{\mathcal{I}_1}$ processes job j using the interval $[b_j, \tau_i)$ while in the interval $[\tau_i, e_j)$, $\overline{OPT}_{\mathcal{I}_1}$ does not process any job. It is clear that $\overline{OPT}_{\mathcal{I}_1}$ executes job j completely within one of its allowed intervals; and none of the jobs execution times overlap. Furthermore, by (1),

$$\text{Speed of } j \text{ in } \overline{OPT}_{\mathcal{I}_1} = \frac{v_j}{\tau_i - b_j} \leq 2\frac{v_j}{e_j - b_j} = 2 \cdot \text{Speed of } j \text{ in } OPT_{\mathcal{I}}.$$

By Proposition 1,

$$E(\overline{OPT}_{\mathcal{I}_1}) = \sum_{j \in \mathcal{J}} E(\overline{OPT}_{\mathcal{I}_1}, j) \leq 2^{\alpha-1} \sum_{j \in \mathcal{J}} E(OPT_{\mathcal{I}}, j) = 2^{\alpha-1}E(OPT_{\mathcal{I}}).$$

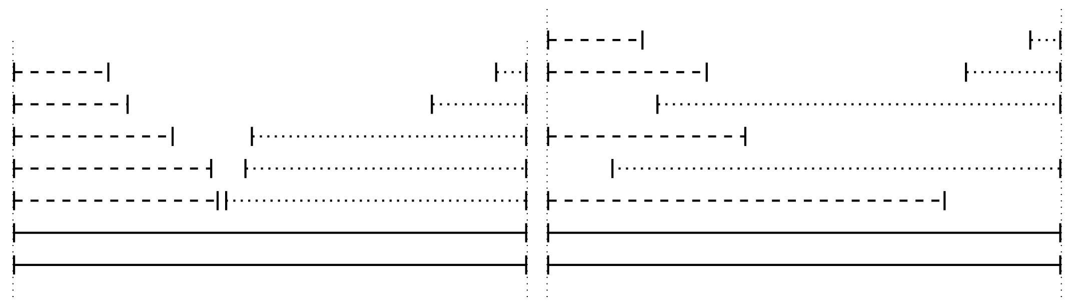
Now the proof follows from the fact that $E(OPT_{\mathcal{I}_1}) \leq E(\overline{OPT}_{\mathcal{I}_1})$. □

Transformation II

In the second transformation, inside each zone $[\tau_{i-1}, \tau_i)$, we define a set of *sublandmarks* to further subdivide the zone into a set of *subzones*. The allowed intervals of a job are (possibly) shortened and then further fragmented by these subzones. We now flesh out the details.

Recall that each job in $\mathcal{J}$ has at most one allowed interval in the zone $[\tau_{i-1}, \tau_i)$. Let $\mathcal{J}' \subseteq \mathcal{J}$ be the subset of the jobs that have exactly one allowed interval in this zone. For simplicity, we assume that the allowed interval of a job $j \in \mathcal{J}'$ is also its first allowed interval $I_{j1} = [r_{j1}, d_{j1})$. We divide the jobs in $\mathcal{J}'$ into three groups.

$$\begin{aligned} j \in A &\quad \text{if } r_{j1} = \tau_{i-1}, d_{j1} < \tau_i; \\ j \in B &\quad \text{if} \tau_{i-1} < r_{j1}, d_{j1} = \tau_i; \\ j \in C &\quad \text{if} r_{j1} = \tau_{i-1}, d_{j1} = \tau_i. \end{aligned}$$



(a) Laminar family case (b) General case.

Figure 2: A zone. The dashed lines represent allowed intervals for jobs of group A, the dotted ones for jobs of group B. Finally, the solid lines represent the allowed intervals for jobs of group C. (a) illustrates a zone after Transformation I. (b) illustrates a zone after Transformation I_1 (see Section 4).

See Figure 2(a) for an illustration. Observe that jobs in group C have their allowed intervals span the entire zone; furthermore, by our assumption that the original instance $\mathcal{I}$ is a laminar instance, the allowed intervals of jobs in group A do not overlap with the allowed intervals of jobs in group B.

Lemma 2. *Let $\mathcal{S}$ be a feasible schedule for instance I_1 and let $\mathcal{J}''$ be the jobs of $\mathcal{J}'$ that are processed within $[\tau_{i-1}, \tau_i)$. Then $\mathcal{S}$ can be transformed into a feasible schedule $\mathcal{S}'$ with the following properties:*

- *$\mathcal{S}'$ consumes no more energy than $\mathcal{S}$.*
- *All jobs in $\mathcal{J}''$ are processed within the zone $[\tau_{i-1}, \tau_i)$.*
- *$\mathcal{S}'$ executes all the jobs of group $A \cap \mathcal{J}''$ before the jobs of group $C \cap \mathcal{J}''$, which in turn are executed before the jobs of group $B \cap \mathcal{J}''$, in $[\tau_{i-1}, \tau_i)$. Moreover, in $\mathcal{S}'$, the jobs of $A \cap \mathcal{J}''$ are processed according to the earliest deadline first principle and the jobs of $B \cap \mathcal{J}''$ according to the earliest release time first principle.*

Proof. This follows from the fact that the allowed intervals of jobs in group C span the entire zone and the fact that jobs j in group A (resp. group C) have the release time $r_{j1} = \tau_{i-1}$ (resp. the deadline $d_{j1} = \tau_i$). ☐

Let d_{Al} be the latest deadline for the jobs in group A and r_{Bf} the earliest release time for the jobs in group B. Let $\lambda_A = \tau_{i-1} + 3/4(d_{Al} - \tau_{i-1})$ and $\lambda_B = \tau_i - 3/4(\tau_i - r_{Bf})$ be two sublandmarks and $[\lambda_A, \lambda_B)$ be a subzone (we will define more sublandmarks inside $[\tau_{i-1}, \lambda_A)$ and $[\lambda_B, \tau_i)$ in a moment). Then

$$\lambda_B - \lambda_A = (\lambda_B - r_{Bf}) + (d_{Al} - \lambda_A) + (r_{Bf} - d_{Al}) =$$
$$\frac{(\tau_i - r_{Bf}) + (d_{Al} - \tau_{i-1})}{4} + r_{Bf} - d_{Al} = \frac{\tau_i - \tau_{i-1}}{4} + \frac{3(r_{Bf} - d_{Al})}{4} \geq \frac{\tau_i - \tau_{i-1}}{4}. \tag{2}$$

Therefore, the subzone $[\lambda_A, \lambda_B)$ is of length at least a fourth of the entire zone. Intuitively speaking, in our proof, this subzone is "reserved" for the jobs in group C. Given any schedule for $\mathcal{I}_1$, even if the processing of the jobs in group C spans the entire zone, we can always increase their processing speed by a factor of 4 and process them entirely within the subzone $[\lambda_A, \lambda_B)$.

Let d_{Af} be the earliest deadline in group A and r_{Bl} the latest release time in group C. Also, let x be the largest positive integer so that $\tau_{i-1} + (d_{Af} - \tau_{i-1})2^{x-1} < \lambda_A$ and y to be the largest positive integer so that $\tau_i - (\tau_i - r_{Bl})2^{y-1} > \lambda_B$. We define the following sublandmarks.

$$\begin{aligned}
&\lambda_A^k = \tau_{i-1} + (d_{Af} - \tau_{i-1})2^{k-1}, && \text{for } 1 \leq k \leq x;\\
&\lambda_A^{x+1} = \lambda_A;\\
&\lambda_B^{y+1} = \lambda_B;\\
&\lambda_B^k = \tau_i - (\tau_i - r_{Bl})2^{k-1}, && \text{for } 1 \leq k \leq y.
\end{aligned}$$

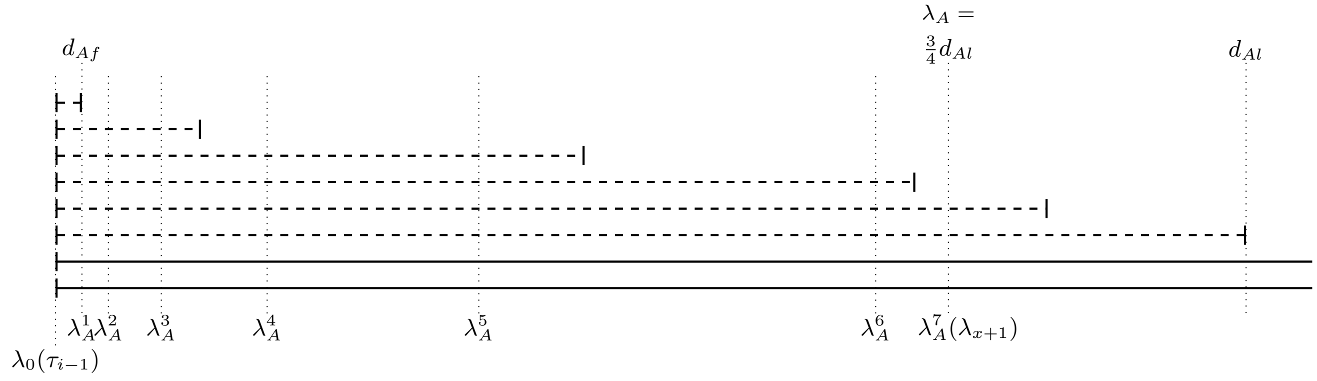


Figure 3: The sublandmarks inside the interval $[\tau_{i-1}, \lambda_A)$.

Notice that in the case that $\tau_{i-1} + (d_{Af} - \tau_{i-1}) = d_{Af} \geq \lambda_A$, we do not define any sublandmark in $[\tau_{i-1}, \lambda_A)$; similarly for $[\lambda_B, \tau_i)$, if $\tau_i - (\tau_i - r_{Bl}) \geq \lambda_B$. (Also note that it is possible that $d_{Af} = d_{Al}$ or $r_{Bf} = r_{Bl}$). Finally, observe that as we assume that all release times and deadlines are integers, there can be only $O(\lceil \log d_{\max} \rceil)$ sublandmarks in each zone.

The sublandmarks $\{\lambda_A^k\}_{k=1}^x$ and $\{\lambda_B^k\}_{k=1}^y$ are used to partition the intervals $[\tau_{i-1}, \lambda_A)$ and $[\lambda_B, \tau_i)$ into subzones respectively. See Figure 3 for an illustration. It can be observed that the sizes of the subzones in $[\tau_{i-1}, \lambda_A)$ grow geometrically, except the first and the last one; similarly, the sizes of the subzones $[\lambda_B, \tau_i)$, except the first and the last one, decrease geometrically. Roughly speaking, in our proof, these subzones will be "reserved" for jobs in group A and B respectively.

We now use the set of sublandmarks $\{\lambda_A^k\}_{k=1}^x \cup \{\lambda_B^k\}_{k=1}^y \cup \lambda_A \cup \lambda_B$ to partition the allowed intervals of all jobs in $\mathcal{J}'$ in a somewhat similar manner to the previous transformation. Suppose there are totally g sublandmarks in the zone $[\tau_{i-1}, \tau_i)$. Let λ_w denote the w-th sublandmark (in increasing order) for $1 \leq w \leq g$. For convenience, let $\lambda_0 = \tau_{i-1}$ and $\lambda_{g+1} = \tau_i$.

We create a new instance $\mathcal{I}_2$ based on $\mathcal{I}_1$ as follows. For a job $j \in \mathcal{J}'$ with the allowed interval $I_{j1} = [r_{j1}, d_{j1})$, let λ_u be the first sublandmark so that $r_{j1} \leq \lambda_u$ and λ_{u+k} be the last sublandmark so that $\lambda_{u+k} \leq d_{j1}$. Then

$$r_{j1} \leq \lambda_u < \lambda_{u+1} < \cdots < \lambda_{u+k} \leq d_{j1}.$$

We replace its allowed interval $I_{j1} = [r_{j1}, d_{j1})$ with a set of allowed intervals $\bigcup_{s=1}^k I_{j1s}$, where

$$I_{j1s} = [\lambda_{u+s-1}, \lambda_{u+s}), \forall s, 1 \leq s \leq k.$$

Note that by this definition, a subzone $[\lambda_{k-1}, \lambda_k)$ becomes an allowed interval of job j in instance $\mathcal{I}_2$ if and only if I_{j1} spans the *entire* interval $[\lambda_{k-1}, \lambda_k)$. In the case that $r_{j1} < \lambda_u$ (this may happen for jobs in $B \cap \mathcal{J}''$) or $d_{j1} > \lambda_{u+k}$ (which may happen for jobs in $A \cap \mathcal{J}''$), $[r_{j1}, \lambda_u)$ or $[\lambda_{u+k}, d_{j1})$ respectively are not allowed intervals.

Lemma 3. *Let $OPT_{\mathcal{I}_1}$ and $OPT_{\mathcal{I}_2}$ denote the optimal schedules for instances $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively. Then $E(OPT_{\mathcal{I}_2}) \leq 4^{\alpha-1} E(OPT_{\mathcal{I}_1})$.*

Proof. We construct a feasible solution $\overline{OPT}_{\mathcal{I}_2}$ for instance $\mathcal{I}_2$ based on the optimal solution $OPT_{\mathcal{I}_1}$ for instance $\mathcal{I}_1$. Let $\mathcal{J}'' \subseteq \mathcal{J}'$ be the subset of jobs that are processed in the zone $[\tau_{i-1}, \tau_i)$ in the schedule $OPT_{\mathcal{I}_1}$. By Lemma 2, we can assume that the jobs in $A \cap \mathcal{J}''$ are processed first, the jobs in $C \cap \mathcal{J}''$ second, followed by the jobs in $B \cap \mathcal{J}''$. We also can assume that jobs in $A \cap \mathcal{J}''$ (resp. $B \cap \mathcal{J}''$) are processed in the order of their increasing deadlines (resp. increasing release times).

Suppose that there exists at least one sublandmark within $[\tau_{i-1}, \lambda_A)$. Recall that we define $\lambda_A^k = \tau_{i-1} + (d_{Af} - \tau_{i-1})2^{k-1}$ for $1 \leq k \leq x$; furthermore, $\lambda_A^{x+1} = \lambda_A$. In the rest of this proof, let $\lambda_k := \lambda_A^k$ for $1 \leq k \leq x+1$. For simplicity, we can assume that $\tau_{i-1} = 0$. By rescaling, let $d_{Af} = 1$.

The following facts follow straightforwardly from the definitions and the assumptions:

$$\begin{aligned}\lambda_k &= 2^{k-1}, \text{for each } 1 \le k \le x;\\ \lambda_{x+1} &= (3/4)d_{Al} \le 2^x.\end{aligned}$$

For convenience, let $\lambda_0 = \tau_{i-1}$. We show how each of the jobs in $A \cap \mathcal{J}''$ can be processed entirely within one of these subzones $[\lambda_{k-1}, \lambda_k)$ when we build the schedule $\overline{OPT}_{\mathcal{I}_2}$. The basic idea is this: treat these subzones as bins with capacity equal to their length, and the jobs $j \in A \cap \mathcal{J}''$ as items whose sizes are their processing time $|OPT_{\mathcal{I}_1}(j)|$ divided by a factor of 4. Each item $j \in A \cap \mathcal{J}''$ can only be put into a bin (subzone) whose interval is entirely contained in j's allowed interval I_{j1}. If this can be done, then we have a new schedule where the jobs in $A \cap \mathcal{J}''$ are feasibly processed, i.e., within their allowed intervals in instance $\mathcal{I}_2$, with processing speeds equal to 4 times their original speeds in $OPT_{\mathcal{I}_1}$.

Let $t_j = |OPT_{\mathcal{I}_1}(j)|$ for all jobs $j \in A \cap \mathcal{J}''$. Moreover, let us divide $A \cap \mathcal{J}''$ into disjoint sets $\mathcal{J}_1 \dot\cup \mathcal{J}_2 \dot\cup \cdots \dot\cup \mathcal{J}_{x+1} \dot\cup \mathcal{J}_{x+2}$ as follows. Job $j \in \mathcal{J}_k$ if its finishing time $e_j(OPT_{\mathcal{I}_1}) \in [\lambda_{k-1}, \lambda_k)$ for $1 \le k \le x+1$. In case that $e_j(OPT_{\mathcal{I}_1}) \in [\lambda_{x+1}, d_{Al})$, let $j \in \mathcal{J}_{x+2}$.

We note that if $j \in \mathcal{J}_k$ for $k \ge 2$, then j can be feasibly processed within any of the subzones $[\lambda_{k'-1}, \lambda_{k'})$ for any $1 \le k' \le k-1$ in instance $\mathcal{I}_2$, since each such subzone $[\lambda_{k'-1}, \lambda_{k'})$ will be one of the allowed intervals of j in $\mathcal{I}_2$ (this follows from the fact that $e_j(OPT_{\mathcal{I}_1}) \ge \lambda_{k-1}$.) Note also that all jobs in $\mathcal{J}_1$ can be feasibly processed within the subzone $[\lambda_0, \lambda_1)$ in instance $\mathcal{I}_2$, since in $\mathcal{I}_1$, all jobs in $A \cap \mathcal{J}''$ have their deadlines at least as late as $d_{Af} = \lambda_1$. These facts are used in the proof of the following claims.

Claim 1. *Suppose that there exists at least one sublandmark within $[\tau_{i-1}, \lambda_A)$. We can process all jobs of $A \cap \mathcal{J}''$ in $\overline{OPT}_{\mathcal{I}_2}$ at four times their speeds in $OPT_{\mathcal{I}_1}$. Moreover, in $\overline{OPT}_{\mathcal{I}_2}$, the following hold:*

1. *Suppose that $x = 1$. Then all jobs in $A \cap \mathcal{J}'' = \mathcal{J}_1 \cup \mathcal{J}_2 \cup \mathcal{J}_3$ are feasibly processed within the subzone $[\lambda_0, \lambda_1)$;*
2. (a) *Suppose that $x \ge 2$. Then all jobs in $\mathcal{J}_1 \cup \mathcal{J}_2$ are feasibly processed within the subzone $[\lambda_0, \lambda_1)$.*
   (b) *Suppose that $x \ge 3$. Then all jobs in $\mathcal{J}_k$ are feasibly processed within the subzone $[\lambda_{k-2}, \lambda_{k-1})$ for $3 \le k \le x$;*
   (c) *Suppose that $x \ge 2$. Then all jobs in $\mathcal{J}_{x+1} \cup \mathcal{J}_{x+2}$ are feasibly processed within the subzones $[\lambda_{x-1}, \lambda_x)$ and/or $[\lambda_x, \lambda_{x+1})$.*

Proof. For (1) suppose that $x = 1$. Then $\lambda_2 = (3/4)d_{Al} \le 2^1 = 2$, implying that $d_{Al} \le 8/3$. We can thus execute all jobs in $A \cap \mathcal{J}''$ with quadrupled speeds within this subzone, since it takes at most $d_{Al}/4 = 2/3$ amount of time. So (1) is proved.

For (2a), first observe that the subzone $[\lambda_0, \lambda_1)$ is of size $2^0 - 0 = 1$. Suppose that $x \ge 2$. As $\sum_{j \in \mathcal{J}_1 \cup \mathcal{J}_2} t_j \le \lambda_2 = 2$, we can thus execute all jobs in $\mathcal{J}_1 \cup \mathcal{J}_2$ with quadrupled speeds within this subzone, since it takes at most $\lambda_2/4 = 0.5$ amount of time.

(2b) can be proved similarly. As $\sum_{j \in \mathcal{J}_k} t_j \le \lambda_k$, we can execute all jobs in $\mathcal{J}_k$ with quadrupled speeds in time at most $\lambda_k/4 = 2^{k-3}$. The subzone $[\lambda_{k-2}, \lambda_{k-1})$ has enough size for it, since $\lambda_{k-1} - \lambda_{k-2} = 2^{k-3}$.

(2c) Consider the following two cases.

Case 1. Suppose that $\mathcal{J}_{x+1} = \emptyset$. There is nothing to prove if $\mathcal{J}_{x+2} = \emptyset$. So assume that $\mathcal{J}_{x+2} \ne \emptyset$. We show below that all jobs in $\mathcal{J}_{x+2}$ can be processed with quadrupled speeds in either $[\lambda_{x-1}, \lambda_x)$ or $[\lambda_x, \lambda_{x+1})$.

Recall that $2^{x-1} = \lambda_x < \lambda_{x+1} = (3/4)d_{Al} \le 2^x$. Therefore,

$$(4/3)2^{x-1} < d_{Al} \le (4/3)2^x.$$

Suppose that $d_{Al} \le 2^x$. We can process all jobs in $\mathcal{J}_{x+2}$ with quadrupled speeds in the subzone $[\lambda_{x-1}, \lambda_x)$, since

$$\lambda_x - \lambda_{x-1} = 2^{x-2} \ge \frac{d_{Al}}{4} \ge \frac{\sum_{j \in \mathcal{J}_{x+2}} t_j}{4}.$$

Next suppose that $d_{Al} = (2^x + k) \le (4/3)2^x$, for some $k > 0$. Then we can similarly quadruple the speeds of all jobs in $\mathcal{J}_{x+2}$ and process them in the subzone $[\lambda_x, \lambda_{x+1})$ because

$$\lambda_{x+1} - \lambda_x = (3/4)d_{Al} - 2^{x-1} = 2^{x-2} + (3/4)k \geq 2^{x-2} + k/4 = \frac{d_{Al}}{4} \geq \frac{\sum_{j \in \mathcal{J}_{x+2}} t_j}{4}.$$

Case 2. Suppose that $\mathcal{J}_{x+1} \neq \emptyset$. Let

$$\lambda_x + t = (3/4)d_{Al} = \lambda_{x+1}. \tag{3}$$

If we can quadruple the speeds of all jobs in $\mathcal{J}_{x+1} \cup \mathcal{J}_{x+2}$ and execute them within the subzone $[\lambda_{x-1}, \lambda_x)$, we then are done. So suppose not. Thus,

$$\lambda_x - \lambda_{x-1} = 2^{x-2} < \frac{\sum_{j \in \mathcal{J}_{x+1} \cup \mathcal{J}_{x+2}} t_j}{4} \leq \frac{d_{Al}}{4},$$

implying that $d_{Al}/4 > 2^{x-2}$, which in turn implies that

$$t > 3 \cdot 2^{x-2} - \lambda_x. \tag{4}$$

We claim that we can process all jobs in $\mathcal{J}_{x+1}$ with quadrupled speeds within the subzone $[\lambda_{x-1}, \lambda_x)$ and all jobs in $\mathcal{J}_{x+2}$ with quadrupled speeds within the subzone $[\lambda_x, \lambda_{x+1})$. To see the former, observe that $\frac{\sum_{j \in \mathcal{J}_{x+1}} t_j}{4} \leq \lambda_{x+1}/4 = 2^{x-2}$, while the subzone $[\lambda_{x-1}, \lambda_x)$ is of size $\lambda_x - \lambda_{x-1} = 2^{x-2}$.

For the latter, we need the following critical observation. *Since $\mathcal{J}_{x+1} \neq \emptyset$, the execution of jobs in $\mathcal{J}_{x+2}$ starts at most as early as λ_x.* This implies that

$$\frac{\sum_{j \in \mathcal{J}_{x+2}} t_j}{4} \leq \frac{d_{Al} - \lambda_x}{4} = \frac{\lambda_x + t}{3} - \frac{\lambda_x}{4} = \frac{\lambda_x}{12} + \frac{t}{3},$$

where the first equality follows from (3).

On the other hand, the size of the subzone $[\lambda_x, \lambda_{x+1})$ is exactly t. Subtracting the upper-bound of $\frac{\sum_{j \in \mathcal{J}_{x+2}} t_j}{4}$ from t we get

$$t - \frac{\lambda_x}{12} - \frac{t}{3} > 2 \cdot 2^{x-2} - \frac{2\lambda_x}{3} - \frac{\lambda_x}{12} = \frac{2^{x-1}}{4} > 0,$$

where the first inequality follows from (4).

Thus we can fit the jobs in $\mathcal{J}_2$ into the subzone $[\lambda_x, \lambda_{x+1})$. The proof follows. □

Claim 2. *Suppose that there is no sublandmark in $[\tau_{i-1}, \lambda_A)$. We can feasibly process all jobs in $A \cap \mathcal{J}''$ in $\overline{OPT}_{\mathcal{I}_2}$ in the subzone $[\tau_{i-1}, \lambda_A)$ so that their speeds are four times their speeds in $OPT_{\mathcal{I}_1}$.*

Proof. Observe that there is no sublandmark within $[\tau_{i-1}, \lambda_A)$ only if $\lambda_A = (3/4)d_{Al} \leq d_{Af} = 1$. In this case, all jobs in $A \cap \mathcal{J}''$ have the subzone $[\tau_{i-1}, \lambda_A)$ as one of its allowed intervals in instance $\mathcal{I}_2$, since $d_{Af} \geq \lambda_A$. Now we can quadruple the speeds of the jobs in $A \cap \mathcal{J}''$ and process them in $\frac{\sum_{j \in A \cap \mathcal{J}''} t_j}{4} \leq d_{Al}/4 = 1/3$ amount of time, which is less than the size of the subzone $[\tau_{i-1}, \lambda_A)$, which is $(3/4)d_{Al} - 0 \geq (3/4)d_{Af} = 3/4$. This completes the proof. □

Claims 1 and 2 imply that the jobs in $A \cap \mathcal{J}''$ can be feasibly processed within the subzones partitioning the interval $[\tau_{i-1}, \lambda_A)$ with quadrupled speeds. By a symmetric argument, in $\overline{OPT}_{\mathcal{I}_2}$, the jobs in $B \cap \mathcal{J}''$ can be feasibly processed within the subzones partitioning the interval $[\lambda_B, \tau_i)$ with quadrupled speeds. Finally, in $\overline{OPT}_{\mathcal{I}_2}$, by equation (2), all jobs in $C \cap \mathcal{J}''$ can be feasibly processed using the subzone $[\lambda_A, \lambda_B)$ with quadrupled speeds. Using Proposition 1, we can thus conclude that $E(\overline{OPT}_{\mathcal{I}_2}) \leq 4^{\alpha-1} E(OPT_{\mathcal{I}_1})$. Now the entire lemma follows from the fact that $E(OPT_{\mathcal{I}_2}) \leq E(\overline{OPT}_{\mathcal{I}_2})$. □

Transformation III

In this section, we transform $\mathcal{I}_2$ into an instance $\mathcal{I}_3$ for *unrelated machine scheduling* with the objective of minimizing the L_α-norm.

> Unrelated Machine Scheduling with L_α-norm objective
> Given m machines $\mathcal{M}$ and n jobs $\mathcal{J}$, where each job j takes p_{ij} processing time on machine i, the goal is to find an assignment A mapping jobs to machines that minimizes $COST(A) = (\sum_{i\in\mathcal{M}}(\sum_{j:A(j)=i} p_{ij})^\alpha)^{1/\alpha}$.

Recall that in $\mathcal{I}_2$, each job $j \in \mathcal{J}$ has a set of allowed intervals, each of which corresponds to a subzone. Let the collection of all subzones be Z and $|z_i|$ denote the length of a subzone $z_i \in Z$. In $\mathcal{I}_3$, each subzone z_i corresponds to a machine $i \in \mathcal{M}$. For each job $j \in \mathcal{J}$ and each subzone z_i, if j does not have an allowed interval corresponding to z_i, then $p_{ij} = +\infty$. If it has such an interval, then let $p_{ij} = \frac{v_j}{|z_i|^{(\alpha-1)/\alpha}}$.

Note that by convexity of the power function, we can assume that a schedule in $\mathcal{I}_2$ processes a set of jobs within a subzone using a uniform speed. The following lemma shows a one-to-one correspondence between a feasible schedule in $\mathcal{I}_2$ and a feasible assignment in $\mathcal{I}_3$.

Lemma 4. *For each assignment A in $\mathcal{I}_3$, a feasible schedule $\mathcal{S}_{\mathcal{I}_2}(A)$ for instance $\mathcal{I}_2$ can be created as follows. If a set of jobs $\mathcal{J}' \subseteq \mathcal{J}$ is assigned to machine $i \in \mathcal{M}$ in A, $\mathcal{S}_{\mathcal{I}_2}(A)$ processes the jobs in $\mathcal{J}'$ with the uniform speed of $\frac{\sum_{j\in\mathcal{J}'} v_j}{|z_i|}$ in subzone z_i. Conversely, given a feasible schedule $\mathcal{S}_{\mathcal{I}_2}(A)$ for instance $\mathcal{I}_2$, let $A_{\mathcal{I}_2}(j)$ denote the subzone in which job j is processed under $\mathcal{S}_{\mathcal{I}_2}(A)$. We can create an assignment A in instance $\mathcal{I}_3$ so that $A(j) = A_{\mathcal{I}_2}(j)$ for all jobs $j \in \mathcal{J}$.*

In both cases, we have $(COST(A))^\alpha = E(\mathcal{S}_{\mathcal{I}_2}(A))$.

Proof. It is clear that $\mathcal{S}_{\mathcal{I}_2}(A)$ and A are a feasible schedule (assignment) for instances $\mathcal{I}_2$ and $\mathcal{I}_3$ respectively. Now the proof follows by observing that

$$E(\mathcal{S}_{\mathcal{I}_2}(A)) = \sum_{z_i\in Z}\left(\frac{\sum_{j:A(j)=i} v_j}{|z_i|}\right)^\alpha |z_i| = \sum_{z_i\in Z}\left(\frac{\sum_{j:A(j)=i} v_j}{|z_i|^{(\alpha-1)/\alpha}}\right)^\alpha =$$

$$\sum_{i\in\mathcal{M}}\left(\sum_{j:A(j)=i} p_{ij}\right)^\alpha = (COST(A))^\alpha.$$

□

Lemma 5. *Let $OPT_{\mathcal{I}_3}$ be an optimal and A be a feasible solution for $\mathcal{I}_3$, such that $COST(A) \le 2COST(OPT_{\mathcal{I}_3})$, i.e., A is a 2-approximate solution for $\mathcal{I}_3$. Let $\mathcal{S}_{\mathcal{I}_2}(A)$ be the corresponding schedule of A and $OPT_{\mathcal{I}_2}$ be the corresponding schedule of $OPT_{\mathcal{I}_3}$ in instance $\mathcal{I}_2$. Then $E(\mathcal{S}_{\mathcal{I}_2}(A)) \le 2^\alpha E(OPT_{\mathcal{I}_2})$.*

Proof. This follows easily from Lemma 4, as

$$E(\mathcal{S}_{\mathcal{I}_2}(A)) = (COST(A))^\alpha \le 2^\alpha(COST(OPT_{\mathcal{I}_3}))^\alpha = 2^\alpha(E(OPT_{\mathcal{I}_2})^{1/\alpha})^\alpha = 2^\alpha E(OPT_{\mathcal{I}_2}).$$

□

We can produce a 2-approximation schedule as stated in Lemma 5 using the algorithm of Azar and Epstein [2]. Combining Lemmas 1, 3, and 5, we prove Theorem 1 for the case of laminar instances.

4 General Instances

For a general instance, our algorithm is similar to what we have done for a laminar instance. Recall that we use the following three transformations for a laminar instance $\mathcal{I}$.

1. Transformation I. Introduce a set of landmarks to transform $\mathcal{I}$ to $\mathcal{I}_1$, where each job's allowed interval is chopped up into a set of allowed intervals.

2. Transformation II. Introduce a set of sublandmarks to transform $\mathcal{I}_1$ to $\mathcal{I}_2$, where each allowed interval of a job is possibly shortened and then further chopped up into a set of allowed intervals.
3. Transformation III. Transform $\mathcal{I}_2$ to $\mathcal{I}_3$, an instance of unrelated machine scheduling.

For a general instance $\mathcal{I}$, Transformations II and III can remain unchanged. But we need to revise Transformation I as follows.

- Transformation I_1. We introduce a sweepline algorithm to define the landmarks.
- Transformation I_2. We shorten a subset of the allowed intervals so that the instance has the same structure as we have had after transformation I when dealing with a laminar instance.

Transformation I_1

We define a set of landmarks using the sweepline algorithm *Sweep* presented in Figure 4. Let $\Psi = \cup_{j \in \mathcal{J}}\{r_j, d_j\}$ be the set of *events*. Note that $\Psi \le 2|\mathcal{J}|$. The order of these events is based on their numerical values, where ties are broken arbitrarily.

Algorithm *Sweep*, sweeps the time horizon from left to right until it meets the first deadline. It then sets this deadline as a landmark. Every job that was seen up to that point gets "removed" and *Sweep* repeats the same procedure.

```
**Algorithm Sweep:**

**Initialization:** Set $\mathcal{J}_c := \emptyset$, and events $\Psi := \cup_{j \in \mathcal{J}}\{r_j, d_j\}$. Assume that the elements $t_i, 1 \le t_i \le |\Psi|$ of $\Psi$ are ordered, and let $T_{\text{land}} := \{0, d_{\max}\}$ be the initial set of landmarks.
**For** $i = 1$ **to** $|\Psi|$ **do**:
    **If** $t_i = r_j$ for some job $j \in \mathcal{J} \backslash \mathcal{J}_c$ **then** $\mathcal{J}_c = \mathcal{J}_c \cup \{j\}$.
    **Else If** $t_i = d_j$ for some $j \in \mathcal{J}_c$ **then**
        add $t_i$ to $T_{\text{land}}$; Set $\mathcal{J}_c := \emptyset$.
```

Figure 4: A sweepline algorithm to define landmarks.

Let $\tau_0 = 0 < \tau_1 < \cdots < \tau_k < \tau_{k+1} = d_{\max}$ be the set of landmarks defined by the above sweepline algorithm. The following lemma follows easily by an inductive argument.

Lemma 6. *Let $\tau_i \in T_{land}$ be the set of landmarks returned by the algorithm Sweep, where τ_i's are ordered increasingly. Within each zone $[\tau_{i-1}, \tau_i)$, for $1 \le i < |T_{land}|$, there exists at least a job $j \in \mathcal{J}$ whose allowed interval I_j is completely contained in such a zone, i.e., $I_j \subseteq [\tau_{i-1}, \tau_i)$.*

We create a new instance $\mathcal{I}_1$ based on $\mathcal{I}$ in the same manner as before. For each job $j \in \mathcal{J}$ with its allowed interval $I_j = [r_j, d_j)$, suppose that

$$\tau_{i-1} \le r_j < \tau_i < \tau_{i+1} < \cdots < \tau_{i+k} < d_j \le \tau_{i+k+1}.$$

Then as in Transformation I, we replace its allowed interval $I_j = [r_j, d_j)$ with a set of allowed intervals $\bigcup_{s=1}^{k+2} I_{js}$, where

$$I_{j1} = [r_j, \tau_i),\ I_{j(k+2)} = [\tau_{i+k}, d_j),\ \text{and } I_{js} = [\tau_{s+i-2}, \tau_{s+i-1}) \text{ for } 2 \le s \le k+1.$$

Lemma 7. *Let $OPT_{\mathcal{I}}$ and $OPT_{\mathcal{I}_1}$ denote the optimal schedules for instances $\mathcal{I}$ and $\mathcal{I}_1$ respectively. Then $E(OPT_{\mathcal{I}_1}) \le 2^{\alpha-1}E(OPT_{\mathcal{I}})$.*

Proof. The proof is almost the same as the proof of Lemma 1. By Lemma 6, we know that no execution $OPT_{\mathcal{I}}(j)$ spans more than one landmark. Using this fact we can build $\overline{OPT}_{\mathcal{I}_1}$ where all jobs are executed within their allowed intervals with at most double their speeds in $OPT_{\mathcal{I}}$. As $\overline{OPT}_{\mathcal{I}_1}$ is feasible for $\mathcal{I}_1$, the proof follows by the fact that $E(OPT_{\mathcal{I}_1}) \le E(\overline{OPT}_{\mathcal{I}_1})$. □

Transformation $\mathbf{I}_2$

In $\mathcal{I}_1$, each job has at most one allowed interval in the zone $[\tau_{i-1}, \tau_i)$. Let $\mathcal{J}' \subseteq \mathcal{J}$ be the set of jobs j that have exactly one allowed interval in $[\tau_{i-1}, \tau_i)$. Assume that $j \in \mathcal{J}'$ has the allowed interval $I_{j1} = [r_{j1}, d_{j1})$. As before, we can divide $\mathcal{J}'$ into three groups A, B and C: (1) $j \in A$, if $r_{j1} = \tau_{i-1}$, $d_{j1} < \tau_i$; (2) $j \in B$ if $\tau_{i-1} < r_{j1}$, $d_{j1} = \tau_i$; and (3) $j \in C$ if $r_{j1} = \tau_{i-1}$, $d_{j1} = \tau_i$.

However, unlike the laminar case, the allowed intervals of jobs in groups A and B can overlap (see Figure 2(b)) in instance $\mathcal{I}_1$. Thus the technique employed in Transformation II breaks down.

To deal with this, we shorten the allowed intervals of jobs in groups A and B so that in the new instance $\mathcal{I}_{1,1}$, the allowed intervals of jobs in group A do not overlap with those of jobs in group B. We argue that the cost of the optimal solution in instance $\mathcal{I}_{1,1}$ does not increase by too much compared to the optimal solution in instance $\mathcal{I}_1$.

We now transform $\mathcal{I}_1$ into $\mathcal{I}_{1,1}$ as follows. For the zone $[\tau_{i-1}, \tau_i)$, let again d_{Al} be the latest deadline in group A and r_{Bf} the earliest release time in group B. Suppose that $j \in \mathcal{J}'$ and its allowed interval is $I_{j1} = [r_{j1}, d_{j1})$.

- If $j \in A$ and $d_{j1} > \tau_{i-1} + \frac{d_{Al} - \tau_{i-1}}{2}$, then replace j's allowed interval with $I'_{j1} = [\tau_{i-1}, \tau_{i-1} + \frac{d_{Al} - \tau_{i-1}}{2})$.
- If $j \in B$ and $r_{j1} < \tau_i - \frac{\tau_i - d_{Bf}}{2}$, then replace j's allowed interval with $I'_{j1} = [\tau_i - \frac{\tau_i - r_{Bf}}{2}, \tau_i)$.

Observe that because Lemma 2 still holds for instances $\mathcal{I}_1$ and $\mathcal{I}_{1,1}$, we can assume that the jobs in group A are processed first, then the jobs in group C, and then jobs in groups B. Jobs in group A (resp. group B) are processed in the order of their increasing deadlines (resp. increasing release times). We show the following lemma.

Lemma 8. *Let $OPT_{\mathcal{I}_1}$ and $OPT_{\mathcal{I}_{1,1}}$ denote the optimal schedules for instances $\mathcal{I}_1$ and $\mathcal{I}_{1,1}$ respectively. Then $E(OPT_{\mathcal{I}_{1,1}}) \leq 2^{\alpha-1} E(OPT_{\mathcal{I}})$.*

Proof. We build a feasible schedule $\overline{OPT}_{\mathcal{I}_{1,1}}$ for instance $\mathcal{I}_{1,1}$ based on $OPT_{\mathcal{I}_1}$ and show that the cost of the former is at most $2^{\alpha-1}$ times larger than the latter.

We construct $\overline{OPT}_{\mathcal{I}_{1,1}}$ for each zone $[\tau_{i-1}, \tau_i)$ separately. For simplicity, assume that $\tau_{i-1} = 0$. Let $\xi_A = \tau_{i-1} + (d_{Al} - \tau_{i-1})/2$ and $\xi_B = \tau_i - (\tau_i - r_{Bf})/2$. Let $\mathcal{J}''$ be the subset of jobs that are processed in this zone $[\tau_{i-1}, \tau_i)$ under $OPT_{\mathcal{I}_1}$.

Similar to the proof of Claim 1, assume that every job j in $\mathcal{J}'' \cap A$ takes $t_j = |OPT_{\mathcal{I}_1}(j)|$ time to finish. We can assume, by Lemma 2, that the jobs in $\mathcal{J}'' \cap A$ are executed by the earliest deadline first principle. In $\overline{OPT}_{\mathcal{I}_{1,1}}$ we process the jobs in $\mathcal{J}'' \cap A$ again according to the earliest deadline first principle but each job j only for time $t_j/2$ (i.e., at twice speed than the one it had in $OPT_{\mathcal{I}_1}$). We need to prove that this modified schedule is feasible.

Observe that $e_j(OPT_{\mathcal{I}_1}) = \sum_{i=1}^{j} t_i$ and $e_j(\overline{OPT}_{\mathcal{I}_{1,1}}) = \sum_{i=1}^{j} \frac{t_i}{2}$, we get that $e_j(\overline{OPT}_{\mathcal{I}_{1,1}}) = \frac{1}{2} e_j(OPT_{\mathcal{I}_1})$ holds for all $1 \leq j \leq |\mathcal{J}'' \cap A|$. Therefore, we claim that all jobs in $A \cap \mathcal{J}''$ are feasibly processed (i.e., before their deadlines) in $\mathcal{I}_{1,1}$, since by our choice of $\xi_A = \tau_{i-1} + (d_{Al} - \tau_{i-1})/2$, their allowed intervals are shortened by at most a factor of 2. By a symmetric argument, all jobs in $\mathcal{J}'' \cap B$ can be feasibly processed within $[\xi_B, \tau_i)$ with their speeds doubled in $\overline{OPT}_{\mathcal{I}_{1,1}}$ for $\mathcal{I}_{1,1}$. Finally in $\overline{OPT}_{\mathcal{I}_{1,1}}$, we can process all jobs in $C \cap \mathcal{J}''$ with their original speeds in $\mathcal{I}_1$, because the jobs in $(A \cup B) \cap \mathcal{J}''$ use only half of their execution times in $\overline{OPT}_{\mathcal{I}_{1,1}}$, and by definition, all jobs in C have their allowed interval spanning the entire zone $[\tau_{i-1}, \tau_i)$.

Finally, as all jobs are processed with at most double speeds in $\overline{OPT}_{\mathcal{I}_{1,1}}$, by Proposition 1,

$$E(\overline{OPT}_{\mathcal{I}_{1,1}}) \leq 2^{\alpha-1} E(OPT_{\mathcal{I}_1}).$$

Now the proof follows by the fact that $E(OPT_{\mathcal{I}_{1,1}}) \leq E(\overline{OPT}_{\mathcal{I}_{1,1}})$. □

Combining Lemmas 7, 8, 3, and 5, we complete the proof of Theorem 1.

References

[1] S. Albers. Energy-efficient algorithms. In Comm. of the ACM, 53(5), 86-96, 2010.

[2] Y. Azar and A. Epstein. Convex programming for scheduling unrelated parallel machines. In *STOC 2005*, 331-337.

[3] N. Bansal, T. Kimbrel and K. Pruhs. Speed scaling to manage energy and temperature. In *J. ACM*, 54:1, 2007

[4] Y. Bartal, S. Leonardi, G. Shallom and R. Sitters. On the value of preemption in scheduling. In *APPROX 2006*, 39-48.

[5] O. Braun and G. Schmid. Parallel processor scheduling with limited number of preemptions. In *SIAM J. Computing*, 32:671-680, 2003.

[6] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. In *IEEE Micro*, 20(6):26-44, 2000.

[7] J.-J. Chen, T.-W. Kuo and H.-I Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. *WADS 2005*, 338-349.

[8] J. Chuzhoy and P. Codenotti. Resource minimization job scheduling. *APPROX 2009*, 70-83.

[9] J. Chuzhoy, R. Ostrovsky and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Math. Oper. Res. 31(4)*, 730-738, 2006.

[10] R.T. Dimpsey and R.K. Iyer. Performance degradation due to multiprogramming and system overheads in real workloads: Case study on a shared memory multiprocessor. In *International Conference on Supercomputing 1990*, 227-238.

[11] Y. Etsion, D. Tsafrir and D.G. Feitelson. Effects of clock resolution on the scheduling of interactive and soft real-time processes. In *SIGMETRICS 1990*, 172-183.

[12] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-Completeness.* W.H. Freeman, 1979.

[13] C. Natarajan, S. Sharma and R.K Iyer. Measurement-based characterization of global memory and network connection, operating system and parallelization overheads: Case study on a shared memory multiprocessor. In *Annual International Symposium on Computer Architecture*, 21:71-80, 1994.

[14] M. Li, B.J. Liu and F.F. Yao. Min-energy voltage allocation for tree-structured tasks. In *J. Combinatorial Optimization*, 11:305-319, 2006.

[15] M. Li and F.F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. on Computing*, 35:658-671, 2005.

[16] F.F. Yao, A.J. Demers and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS 1995*, 374-382.