Noname manuscript No.
(will be inserted by the editor)

Speed Scaling for Maximum Lateness

Evripidis Bampis · Dimitrios Letsios · Ioannis Milis · Georgios Zois

the date of receipt and acceptance should be inserted later

Abstract We consider the power-aware problem of scheduling non-preemptively a set of jobs on a single speed-scalable processor so as to minimize the maximum lateness, under a given budget of energy. In the offline setting, our main contribution is a combinatorial polynomial time algorithm for the case in which the jobs have common release dates. In the presence of arbitrary release dates, we show that the problem becomes strongly $\mathcal{NP}$-hard. Moreover, we show that there is no $O(1)$-competitive deterministic algorithm for the online setting in which the jobs arrive over time. Then, we turn our attention to an *aggregated* variant of the problem, where the objective is to find a schedule minimizing a linear combination of maximum lateness and energy. As we show, our results for the *budget* variant can be adapted to derive a similar polynomial time algorithm and an $\mathcal{NP}$-hardness proof for the aggregated variant in the offline setting, with common and arbitrary release dates respectively. More interestingly, for the online case, we propose a 2-competitive algorithm.

Keywords Energy efficiency · Speed scaling · Scheduling · Maximum lateness

1 Introduction

In classical scheduling an important measure of the *Quality of Service* (QoS) of a schedule is the maximum lateness [8]. Every job, among other characteristics, is associated with a due date and

A preliminary version of this paper has been published in the Proceedings of the 18th Annual International Computing and Combinatorics Conference (COCOON 2012).

E. Bampis and D. Letsios were partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010, and by GDR-RO of CNRS. I. Milis was partially supported by the project THALES-ALGONOW co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program “Education and Lifelong Learning”. G. Zois was supported by Heracleitus II program co-financed by the European Social Fund (ESF) and Greek national funds, through the Operational Program “Education and Lifelong Learning”.

E. Bampis
Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
E-mail: Evripidis.Bampis@lip6.fr

D. Letsios
Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
E-mail: Dimitrios.Letsios@lip6.fr

I. Milis
Dept. of Informatics, Athens University of Economics and Business, Greece
E-mail: milis@aueb.gr

G. Zois
Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France and
Dept. of Informatics, Athens University of Economics and Business, Greece
E-mail: Georgios.Zois@lip6.fr

the *lateness* of a job, with respect to a particular schedule, is defined as the difference of the job's completion time minus its due date, while the *maximum lateness* is computed as the maximum over all jobs. In this paper, we propose to optimize this QoS objective in the context of power management, where the operating system may change the speed of the processor(s) in order to save energy. In general, high processor speeds imply high performance with respect to the QoS criterion (here the maximum lateness) at the price of high energy consumption.

Formally, an instance of our problem consists of a set of n jobs $J = \{1, 2, \ldots, n\}$, where every job i is associated with a release date r_i, a work w_i and a delivery time q_i, that have to be executed non-preemptively on a single speed-scalable processor. Note that in this setting, where jobs are associated with delivery times instead of deadlines, different jobs may be delivered simultaneously. For a given schedule the lateness of job i is defined as $L_i = C_i + q_i$, where C_i is the completion time of job i and the maximum lateness is defined as $L_{\max} = \max_{1 \leq i \leq n}\{L_i\}$. Jobs that attain the maximum lateness in a schedule are referred as *critical* jobs.

At a given time t, if a processor runs at speed s, then its power consumption is $P(s) = s^{\alpha}$, where $\alpha > 2$ is a constant. By integrating the power over time we can compute the processor's energy consumption. That is, if a processor operates at a constant speed s, it executes an amount of work w in w/s time units and consumes an amount of energy $E = ws^{\alpha-1}$.

As maximum lateness minimization and energy savings are conflicting objectives, we consider two variants: In the, so called, *budget variant*, we aim in minimizing $L_{\max} = \max_{i \in J}\{L_i\}$ for a given budget of energy. Using the classical three field notation [10], we denote such a problem by $S1 \mid r_i \mid L_{\max}(E)$, where $S1$ denotes a single speed scalable processor. In the second approach, that we call *aggregated variant*, our objective is to minimize a linear combination of maximum lateness and energy, that is $S1 \mid r_i \mid L_{\max} + \beta E$, where $\beta \geq 0$ is a given parameter that specifies the relative importance of energy versus maximum lateness (see [4] for a motivation of the aggregated approach).

In this context, a schedule σ has to specify for every job the time interval during which it is executed as well as its speed over time. It is well known, e.g. [16], that there is an optimal schedule where each job i is executed at a constant speed; this is a consequence of the convexity of speed-to-power function.

Related work and our results. Yao, Demers and Shenker, in their seminal paper [16] proposed an optimal polynomial time algorithm for finding a feasible *preemptive* schedule on a single processor for a set of jobs with release dates and deadlines minimizing the energy used. They also proposed two online algorithms for the same problem (OA and AVR).

Bunde [6] studied the budget variant of the *non-preemptive* makespan minimization problem for the single-processor case and the multiple processor case with jobs of unit work. He also proved the $\mathcal{NP}$-hardness of the multiprocessor case whenever the jobs have arbitrary works. Pruhs et al. [14] studied the budget variant of the *non-preemptive* multiprocessor makespan minimization problem in the presence of precedence constraints, and proposed an approximation algorithm. They also gave a PTAS for the case with no precedence constraints.

Albers et al. [3] were the first to consider an aggregated variant for a power-aware scheduling problem by studying online and offline versions of the *non-preemptive* problem of minimizing the sum of flow times of the jobs plus energy, with jobs of unit work. The flow time of a job is defined as the difference between its completion time and its release date. It has to be noticed that Pruhs et al. [13] have studied the budget variant of this problem. Bansal et al. [4] proved that there is no $O(1)$-competitive algorithm, for the budget variant, even if all jobs have unit works.

The interested reader may find recent reviews on power-aware scheduling in [1,2].

In this paper we consider the maximum lateness criterion in the power-aware context. For the budget variant we propose an optimal algorithm for the *non-preemptive* single processor case with common release dates, while in Section 3 we prove that the problem, in the presence of release dates becomes strongly $\mathcal{NP}$-hard and it does not admit any $O(1)$-competitive deterministic algorithm. In Section 4, we move to the aggregated variant, and we give an optimal algorithm for the single processor problem with common release dates and a strongly $\mathcal{NP}$-hardness proof for arbitrary release dates. Moreover, we propose a 2-competitive algorithm for the latter case.

2 Budget variant with common release dates

In this section we present a polynomial-time algorithm for the $S1 \mid \mid L_{max}(E)$ problem. Our algorithm is based on a number of structural properties of an optimal schedule, deduced by formulating our problem as a convex program and applying the KKT (Karush, Kuhn, Tucker) conditions.

2.1 General form of KKT conditions

Next, we describe the general form of the KKT conditions for convex programs (see e.g., [5]). Assume that we are given the following convex program:

$$\begin{aligned} &\min f(x) && \\ &g_i(x) \leq 0, && 1 \leq i \leq q \\ &h_j(x) = 0, && 1 \leq j \leq r \\ &x \in \mathbf{R}^n && \end{aligned}$$

Suppose that the program is strictly feasible, i.e. there is a point x such that $g_i(x) < 0$ and $h_j(x) = 0$ for all $1 \leq i \leq q$ and $1 \leq j \leq r$, where all functions g_i and h_j are differentiable at x. Let λ_i and μ_j be the dual variables associated to the constraints $g_i(x) \leq 0$ and $h_j(x) = 0$, respectively. The Karush-Kuhn-Tucker (KKT) conditions are:

$$g_i(x) \leq 0, \qquad 1 \leq i \leq q \tag{1}$$

$$h_j(x) = 0, \qquad 1 \leq j \leq r \tag{2}$$

$$\lambda_i \geq 0, \qquad 1 \leq i \leq q \tag{3}$$

$$\lambda_i g_i(x) = 0, \qquad 1 \leq i \leq q \tag{4}$$

$$\nabla f(x) + \sum_{i=1}^{q} \lambda_i \nabla g_i(x) + \sum_{j=1}^{r} \mu_j \nabla h_j(x) = 0 \tag{5}$$

KKT conditions are necessary and sufficient for solutions $x \in \mathbf{R}^n$, $\lambda \in \mathbf{R}^q$ and $\mu \in \mathbf{R}^r$ to be primal and dual optimal, where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_q)$ and $\mu = (\mu_1, \mu_2, \ldots, \mu_r)$. We refer to the conditions (1) and (2) as primal feasible, to the (3) as dual feasible, to the (4) as complementary slackness and to the (5) as stationarity conditions, respectively.

2.2 A convex programming formulation

A convex programming formulation of our problem stems from two basic properties of an optimal schedule. First, because of the convexity of the speed-to-power function, each job i runs at a constant speed s_i. Second, jobs are scheduled according to the EDD (Earliest Due Date First) rule, or equivalently in non-increasing order of their delivery times; this can be easily shown by a standard exchange argument. Hence, we propose the following formulation where all jobs are considered to be released at time zero and numbered according to the EDD order:

$$\begin{aligned}
&\min L \\
&C_i + q_i \leq L, && 1 \leq i \leq n && (6)\\
&\frac{w_1}{s_1} \leq C_1, && && (7)\\
&C_{i-1} + \frac{w_i}{s_i} \leq C_i, && 2 \leq i \leq n && (8)\\
&\sum_{i=1}^{n} w_i s_i^{\alpha-1} \leq E && && (9)\\
&L, C_i, s_i \geq 0, && 1 \leq i \leq n && (10)
\end{aligned}$$

Our objective is to minimize the maximum lateness, L, among all feasible schedules. Constraints (6) ensure that the lateness of each job is at most L, constraints (7) and (8) enforce the jobs to be scheduled according to the EDD rule in non-overlapping time intervals, constraint (9) does not allow to exceed the given energy budget E and constraints (10) ensure that the maximum lateness, the completion times and the speeds of jobs are non-negative. Constraint (9), for $\alpha > 2$, and constraints (7) and (8) are convex, while constraints (6) and (10) and the objective function are linear. Thus, our mathematical program is indeed convex.

This convex program already implies a polynomial algorithm for our problem, as convex programs can be solved to arbitrary precision by the Ellipsoid algorithm [12]. Since the Ellipsoid algorithm is rather impractical, we will exploit this convex program to derive a fast combinatorial algorithm.

2.3 Properties of an optimal schedule

In what follows we deduce a number of structural properties of an optimal schedule by applying the KKT conditions to the above convex program.

Lemma 1 *For the maximum lateness problem with an energy budget E, the following properties are necessary and sufficient for optimality of a feasible schedule.*
(i) Each job i runs at a constant speed s_i.
(ii) Jobs are scheduled according to the EDD rule.
(iii) There are no idle periods in the schedule.
(iv) The last job is critical, i.e., $L_n = L_{max}$.
(v) Every non-critical job i has equal speed with the job $i+1$, i.e., $s_i = s_{i+1}$.
(vi) Jobs are executed in non-increasing speeds, i.e., $s_i \geq s_{i+1}$.
(vii) All the energy budget is consumed.

Proof In order to apply the KKT conditions to the convex program, we associate to each set of constraints from (6) up to (9), dual variables $\beta_i, \gamma_1, \gamma_i, \delta$, respectively. W.l.o.g. the variables L, C_i and s_i are positive and, by the complementary slackness conditions, the dual variables associated to the constraints (10) are equal to zero.

Stationarity conditions give that

$$\nabla L + \sum_{i=1}^{n} \beta_i \nabla(C_i + q_i - L) + \gamma_1 \nabla(\frac{w_1}{s_1} - C_1)$$
$$+ \sum_{i=2}^{n} \gamma_i \nabla(C_{i-1} + \frac{w_i}{s_i} - C_i) + \delta \nabla(\sum_{i=1}^{n} w_i s_i^{a-1} - E) = 0 \;\Rightarrow$$
$$(1 - \sum_{i=1}^{n} \beta_i)\nabla L + \sum_{i=1}^{n-1} (\beta_i - \gamma_i + \gamma_{i+1})\nabla C_i$$
$$+(\beta_n - \gamma_n)\nabla C_n + \sum_{i=1}^{n} (-\gamma_i w_i s_i^{-2} + (a-1)\delta w_i s_i^{a-2})\nabla s_i = 0$$

Equivalently, we obtain the following equalities.

$$\sum_{i=1}^{n} \beta_i = 1 \tag{11}$$
$$\beta_i = \gamma_i - \gamma_{i+1} \; 1 \le i \le n-1 \tag{12}$$
$$\beta_n = \gamma_n \tag{13}$$
$$(\alpha - 1)\delta = \frac{\gamma_i}{s_i^{\alpha}} \quad 1 \le i \le n \tag{14}$$

The complementary slackness conditions give that

$$\beta_i(C_i + q_i - L) = 0 \; 1 \le i \le n \tag{15}$$
$$\gamma_1(\frac{w_1}{s_1} - C_1) = 0 \tag{16}$$
$$\gamma_i(C_{i-1} + \frac{w_i}{s_i} - C_i) = 0 \; 2 \le i \le n \tag{17}$$
$$\delta\left(\sum_{i=1}^{n} w_i s_i^{\alpha-1} - E\right) = 0 \tag{18}$$

First, we will show that the properties are necessary for optimality. That is, there is always an optimal schedule satisfying them.

(i)-(ii) They have been already discussed above.

(iii) First, note that $\delta \neq 0$. If $\delta = 0$ then by (14), we get that $\gamma_i = 0$ for each $1 \le i \le n$. This, combined with (12) and (13) yields that $\sum_{i=1}^{n} \beta_i = 0$, which is a contradiction because of (11). Since $\delta \neq 0$, we get by (14) that $\gamma_i \neq 0$ for each $1 \le i \le n$. Then, equations (16) and (17) give that there is no idle time in any optimal schedule since $C_1 = \frac{w_1}{s_1}$ and $C_i = C_{i-1} + \frac{w_i}{s_i}$, for $2 \le i \le n$, respectively.

(iv) Since $\delta \neq 0$, by (14), it follows that $\gamma_n \neq 0$ and finally, because of (13), $\beta_n \neq 0$. So, the last job to finish is always a critical job, by (15).

(v) Note that for every non-critical job i, it holds that $C_i + q_i < L$ and (15) implies that $\beta_i = 0$ for every such job. Hence, if a job i is non-critical $\beta_i = 0 \Rightarrow \gamma_i = \gamma_{i+1} \Rightarrow s_i = s_{i+1}$, by (12) and (14), respectively.

(vi) By the dual feasibility conditions and the equations (12) and (14) we get, respectively, that $\beta_i \ge 0 \Rightarrow \gamma_i \ge \gamma_{i+1} \Rightarrow s_i \ge s_{i+1}$. Thus, the jobs are executed with non-increasing speeds.

(vii) If the energy budget is not entirely consumed, then by (18), $\delta = 0$, which is a contradiction, since, as we have already proved, $\delta \neq 0$.

Next, we will show that the properties are also sufficient for optimality. That is, any feasible schedule satisfying them must be optimal. In order to show this, it suffices to prove that, given any feasible schedule satisfying the properties, we can always give values to the dual variables such that the KKT conditions are satisfied.

Consider a feasible schedule and let s_i and C_i be the speed and the completion time of the job i, $1 \leq i \leq n$, respectively. Moreover, let L be the maximum lateness of the schedule. We give values to the dual variables as follows.

$$\delta = \frac{1}{(\alpha-1)s_1^\alpha}$$
$$\gamma_i = \frac{s_i^\alpha}{s_1^\alpha}, \quad 1 \leq i \leq n$$
$$\beta_i = \frac{s_i^\alpha - s_{i+1}^\alpha}{s_1^\alpha}, 1 \leq i \leq n-1$$
$$\beta_n = \frac{s_n^\alpha}{s_1^\alpha}$$

We, now, observe that these values of the dual variables together with the values of the primal variables satisfy the KKT conditions.

Note that

$$\sum_{i=1}^{n} \beta_i = \sum_{i=1}^{n} \frac{s_i^\alpha - s_{i+1}^\alpha}{s_1^\alpha} = \frac{s_1^\alpha}{s_1^\alpha} = 1$$
$$\beta_i = \frac{s_i^\alpha - s_{i+1}^\alpha}{s_1^\alpha} = \frac{s_i^\alpha}{s_1^\alpha} - \frac{s_{i+1}^\alpha}{s_1^\alpha} = \gamma_i - \gamma_{i+1} \; 1 \leq i \leq n-1$$
$$\beta_n = \frac{s_n^\alpha}{s_1^\alpha} = \gamma_n$$
$$(\alpha-1)\delta = \frac{1}{s_1^\alpha} = \frac{s_i^\alpha}{s_1^\alpha}\frac{1}{s_i^\alpha} = \frac{\gamma_i}{s_i^\alpha} \quad 1 \leq i \leq n$$

So the stationarity conditions are satisfied.

Consider now a job i, $1 \leq i \leq n$. If i is critical, then $C_i + q_i = L$. Else, by property (v) we have that, for $1 \leq i \leq n-1$,

$$s_i = s_{i+1} \Leftrightarrow \frac{s_i^\alpha}{s_1^\alpha} = \frac{s_{i+1}^\alpha}{s_1^\alpha} \Leftrightarrow \beta_i = 0$$

Thus, equation (15) is satisfied. By property (iii), we have that $C_1 = \frac{w_1}{s_1}$ and $C_i = C_{i-1} + \frac{w_i}{s_i}$, for $2 \leq i \leq n$. Therefore, equations (16) and (17) are also satisfied. Furthermore, by property (vii), all the energy budget is consumed and the equation (18) holds. Hence, the complementary slackness conditions are satisfied.

Finally, in order to complete our proof, it remains to show that the values of all the dual variables are non-negative. The only case for which this is not straightforward, is for the values of variables β_i, for $1 \leq i \leq n-1$. But, it must be the case that $\beta_i \geq 0$ for all $1 \leq i \leq n-1$, because of the property (vi) and the theorem follows.

We refer to any schedule satisfying the properties of Lemma 1 as a *regular* schedule. By Lemma 1, every optimal schedule is regular and vice versa; however, there might be feasible, but not optimal, non-regular schedules. By (i, j) we denote a sequence of consecutive jobs $i, i+1, \ldots, j$. Any regular schedule can be partitioned into *groups* of jobs, of the form (i, j), where the jobs $i-1$ and j are critical and the jobs $i, i+1, \ldots, j-1$ are not. By Lemma 1(v), all jobs of such a group are executed at the same speed. We denote this common speed by s_j and the total amount of work of jobs in (i, j) by $w(i, j) = \sum_{k=i}^{j} w_k$. Then, the next proposition follows easily from Lemma 1.

Proposition 1 *Let i, j, be two consecutive critical jobs of a regular schedule. The speed of each job in the group $(i+1, j)$ equals to $s_j = \frac{w(i+1,j)}{q_i - q_j}$.*

Proof Assume without loss of generality that i completes before j. Since i and j are both critical, they attain equal maximum latnesses, i.e. $L_i = L_j$. Moreover, in any regular schedule, by Lemma 1(iv), there is no idle period between jobs $i, i+1, \ldots, j$. Furthermore, all jobs $i+1, i+2, \ldots, j-1$

are non-critical and, by Lemma 1(vi), they are all executed with speed equal to that of job j. Hence, we get, respectively, that

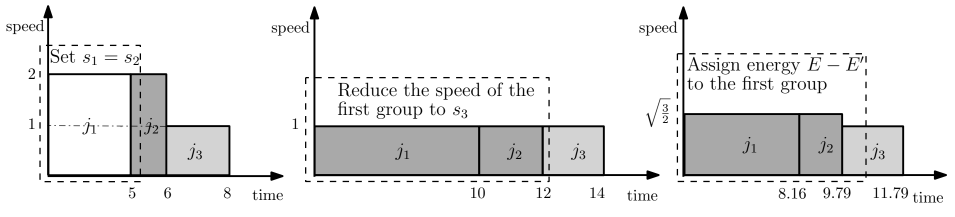
$$L_i = L_j \Rightarrow C_i + q_i = C_j + q_j \Rightarrow \sum_{k=1}^{i} \frac{w_k}{s_k} + q_i = \sum_{k=1}^{j} \frac{w_k}{s_k} + q_j \Rightarrow s_j = \frac{w(i+1,j)}{q_i - q_j}.$$

Clearly, given that $L_i = L_j$ and $C_i < C_j$, it must be the case that $q_i \neq q_j$.

2.4 An optimal combinatorial algorithm

So far, by proving that the properties of a regular schedule are necessary and sufficient for optimality, we have derived a clear image of the structure of an optimal schedule for the $S1 \mid \mid L_{max}(E)$ problem. Next, we propose Algorithm BUD which constructs such a schedule in polynomial time. Note that a regular schedule is fully specified by the speeds of the jobs. The rough idea of our algorithm is the following: First, it constructs a preliminary schedule by finding groups of jobs running in non-increasing speeds without taking care of the energy consumption. Second, the algorithm manages the energy consumption w.r.t. the energy budget E and determines the final speeds of all jobs. Let E' be the energy consumption of the current schedule at any point of the execution of the algorithm.

Algorithm BUD needs the jobs to be ordered/numbered according to the EDD rule and an initial sorting step is required. Once this step is performed, it starts from job n which is always a critical job and considers all jobs but the first, in reverse order. When a job i, $2 \leq i \leq n$, is considered for the first time, its speed s_i is set according to Proposition 1, assuming that jobs $i-1$ and i are critical. If $s_i \geq s_j$, for $i+1 \leq j \leq n$, then s_i is called *eligible* speed and it is assigned to job i; by definition, the speed $s_n = \frac{w_n}{q_{n-1}-q_n}$ is considered to be eligible. If speed s_i is not eligible, i is a non-critical job and it is merged with the $(i+1)$'s group. More specifically, if c is the last job of this group, then the speeds of jobs $i, i+1, \ldots, c$ are calculated by applying Proposition 1, assuming that $i-1$ and c are critical while $i, i+1, \ldots, c-1$ are not. Next, the algorithm examines whether the new value of s_i is eligible. If this is the case, then it considers the job $i-1$. Otherwise, a further merging of the i's group with the $(c+1)$'s group, is performed, as before. That is, if c' is the last job of the $(c+1)$'s group, all jobs $i, i+1, \ldots, c'$ are assigned the same speed assuming that jobs $i-1$ and c' are critical, while $i, i+1, \ldots, c'-1$ are not. This speed, according to the Proposition 1, is equal to $s_{c'} = \frac{w(i,c')}{q_{i-1}-q_{c'}}$. Note that the job c is no longer critical in this case. This merging procedure is repeated until job i is assigned an eligible speed. In a degenerate case, jobs $i, i+1, \ldots, n$ are merged into one group. When the algorithm has assigned an eligible speed to all jobs $2, 3, \ldots, n$, it sets $s_1 = s_2$ and its first part completes. Note that s_1 becomes also eligible. An example of the first part of our algorithm is given in Figure 1(i).



(i) $L_{max} = 10,\ E' > E,\ E' = 50$ (ii) $L_{max} = 16,\ E' < E,\ E' = 14$ (iii) $L_{max} = 13.79,\ E' = 20$

Fig. 1 The execution of Algorithm BUD for an instance of 3 jobs, without release dates, works 10, 2, 2, delivery times 5, 4, 2, $\alpha = 3$ and $E = 20$.

Next, Algorithm BUD takes into account the available budget of energy E. If $E - E' \geq 0$, the current schedule's energy consumption does not exceed the budget of energy, and the surplus

$E - E'$ is assigned to the first job. Otherwise, the current schedule is regular, except that it consumes an amount of energy greater than E. Then, the algorithm reduces the consumed energy until it becomes equal to E. In fact, it decreases the speed of the first group, by merging subsequent groups with it, if necessary. This merging procedure is different from the one of the first part of the algorithm and it is as follows: let i be the critical job of maximal index with $s_i = s_1$ in the current schedule. Observe that $s_i > s_{i+1}$. The algorithm sets the speed of jobs $1, 2, \ldots, i$ equal to s_{i+1}. This causes a reduction to E' and there are two cases to distinguish: either $E' \leq E$ or $E' > E$. In the first case, the algorithm adds an amount of energy $E - E'$ to jobs $1, 2, \ldots, i$ by increasing their speeds uniformly, i.e. so that they are all executed with the same speed. In the second case, at least one further merging step has to be performed. When the algorithm terminates, it is obvious that $E' = E$. For an example of the second part of our algorithm see Figures 1(ii) and 1(iii).

Algorithm BUD

```
1: Sort the jobs according to the EDD order.
2: for $i = n$ to 2 do
3:     Set $s_i$ assuming that $i$ and $i-1$ are critical.
4:     while $s_i$ is not eligible do
5:         Merge the $i$'s group with the next group.
6: Set $s_1 = s_2$
7: Let $E'$ be the current energy consumption.
8: if $E > E'$ then
9:     Assign energy $E - E'$ to job 1.
10: else
11:     while $E < E'$ do
12:         Set the speed of the first group equal to the speed of the following group.
13:         Update $E'$.
14:         if $E < E'$ then
15:             Merge the first group with the next one.
16:     Assign $E - E'$ energy uniformly to the first group.
```

Theorem 1 *Algorithm BUD is optimal for the $S1 \mid \mid L_{max}(E)$ problem.*

Proof We shall prove that the algorithm satisfies the properties of Lemma 1, i.e., it produces a regular schedule. For convenience, we distinguish two parts in the algorithm: *Part I*, corresponding to lines 1-6 and *Part II*, corresponding to lines 7-16, respectively.

Property (i)-(ii): The algorithm gives a single constant speed to each job and keeps their initial EDD order.

Property (iii): In Part I, the speeds of jobs are assigned according to Proposition 1. Specifically, the algorithm fixes two consecutive critical jobs i and j, $i < j$, with, potentially, some non-critical jobs between them. Then the speed of the non-critical jobs and the one of the critical job j is defined such that there is no idle period between the jobs. In Part II, no idle period is added between any jobs.

Property (iv) - (v): When the speed of job n is initialized, this is done by assuming that it is critical. Next, consider the current schedule just after the completion of Part I. This schedule can be partitioned into sequences of jobs, $a+1, a+2, \ldots, b$, with $a \geq 1$, such that the jobs of each sequence are executed with the same speed which has been assigned by applying Proposition 1, assuming that the jobs a and b are critical. In fact, jobs a and b attain equal lateness. In order for such a sequence to be a group, we should also prove that all but the last jobs are non-critical while the last job is critical.

Let $a+1, a+2, \ldots, b$ be a sequence of jobs. We claim that $L_i < L_b$, for $a+1 \leq i \leq b-1$. Assume, by contradiction, that there exists a job j, where $a+1 \leq j \leq b-1$, such that $L_j \geq L_b$, or equivalently, $q_j - q_b \geq \sum_{i=j+1}^{b} \frac{w_i}{s_b}$. Since the last job of a sequence attains equal lateness with the last job of the sequence that follows, we have that $L_a = L_b$. This yields that $q_a - q_b = \sum_{i=a+1}^{b} \frac{w_i}{s_b}$. Therefore, $q_a - q_j \leq \sum_{i=a+1}^{j} \frac{w_i}{s_b}$.

Obviously, for any job i, $a+1 \leq i \leq b-1$, we must have a speed $s_i > \frac{w_i}{q_{i-1}-q_i}$, since otherwise, it wouldn't have been merged with another group. That is, $q_{i-1} - q_i > \frac{w_i}{s_i}$. If we sum the last inequalities for $a+1 \leq i \leq j$, we get that $q_a - q_j > \sum_{i=a+1}^{j} \frac{w_i}{s_b}$, a contradiction.

At this point, we have showed that when Part I completes, if a job i, $2 \leq i \leq n$, is critical, then it must be the right extremity of a sequence. Moreover, among all jobs $2, 3, \ldots, n$, the last jobs of all sequences, including job n, attain equal lateness and the remaining jobs attain smaller lateness. In addition, job 1 attains equal lateness with the last job of the sequence that follows. Recall that, at this point, we set $s_1 = s_2$. Job 1 would have equal lateness with the last job of the sequence that follows for any $s_1 > 0$ since the speed of the second group is set by applying Proposition 1, assuming that 1 is critical. So, at the end of Part I, job 1, job n and every last job of a sequence are critical. Therefore, after Part I finishes, Properties (iv) and (v) hold.

In Part II, if no merging step is performed, then the processing time of job 1 is decreased by some $t \geq 0$ and its lateness decreases by t, while the processing times and speeds of the other jobs are not modified. So, the lateness of every other job also decreases by t. Hence, the Properties (iv) and (v) hold.

If at least one merging step is performed, then the speed of the jobs in the first group decreases and their processing time increases. Then, in the first group, every non-critical job i has equal speed with the job $i+1$ that follows, while the speeds of the jobs in other groups remain unchanged. Now, let t_i be the total increase in the processing time of job i, $1 \leq i \leq n$. Note that this quantity is positive only for jobs belonging to the first group of the current schedule. Then, the lateness of any job i, $1 \leq i \leq n$, increases by $\sum_{j=1}^{i} t_j$; if c_1 is the critical job of the first group, it remains critical after the merging step since its lateness and the lateness of every other job that follows, increase by the same quantity, equal to $\sum_{j=1}^{c_1} t_j$. Note, that if a further merging step is performed, we consider the first two groups as one group. Moreover, the lateness of any job increases by no more than the increase of the lateness of job n, and thus, in the final schedule, job n remains critical and Property (iv) holds. Furthermore, each non-critical job has equal speed with the job that follows and Property (v) holds as well.

Property (vi): At the end of Part I, the speeds of jobs are non-increasing since otherwise, a merging step would be performed. Moreover, during Part II, no speed of a job becomes less than the speed of a subsequent job.

Property (vii): Recall that E' is the total energy consumed when Part I completes. If E' is less than the energy budget, then the energy of the first job increases until the schedule consumes exactly E units of energy, while if E' is greater than the energy budget E, then the energy consumption of the schedule is gradually decreased until it becomes equal to E.

Let us now consider the complexity of the algorithm. Initially, jobs are sorted according to the EDD rule in $O(n \log n)$ time. The first part of the algorithm may take $O(n^2)$ time since each merging step takes $O(n)$ time and there can be $O(n)$ merging steps. Also, the algorithm's second part takes $O(n^2)$ time since the speed of each job may change at most $O(n)$ times. Therefore, the overall complexity of the algorithm is $O(n^2)$.

3 Budget variant with arbitrary release dates

We now consider the budget variant of the maximum lateness problem, where the jobs have arbitrary release dates, i.e., $S1 \mid r_j \mid L_{max}(E)$ and we show that it becomes strongly $\mathcal{NP}$-hard. Moreover, we show that there is no $O(1)$-competitive algorithm for its online version, even when all jobs have unit works.

3.1 NP-hardness

We reduce 3-PARTITION to the $S1 \mid r_j \mid L_{max}(E)$ problem. 3-PARTITION problem is a well known $\mathcal{NP}$-hard [7] problem where, we are given a positive integer B and a set of $3n$ positive integers $A = \{a_1, a_2, \ldots, a_{3n}\}$, where $B/4 < a_i < B/2$ and $\sum_{a_i \in A} a_i = nB$, and we ask if

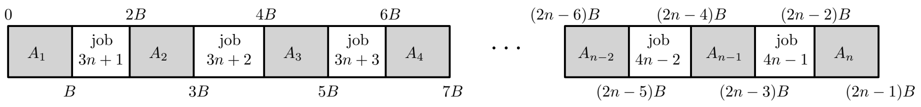


Fig. 2 A feasible schedule σ for $S1 \mid r_j \mid L_{max}(E)$ that attains maximum lateness equal to $L_{max} = (2n-1)B$.

there exists a partition of A into n disjoint sets $A_1, A_2 \dots, A_n$ such that, for each $1 \leq k \leq n$, $\sum_{a_i \in A_k} a_i = B$.

Our reduction is inspired by the $\mathcal{NP}$-hardness proof for the classical $1 \mid r_j \mid L_{max}$ problem [11], where we are given a set of jobs with each job i having a release date r_i, a due date d_i and a processing time p_i and we seek a schedule minimizing the maximum lateness; note that, the feasibility version of this later problem is also known as the SEQUENCING WITHIN INTERVALS problem [7].

The $1 \mid r_j \mid L_{max}$ problem can be viewed as a variant of our problem where the speed of each job is part of the instance. Specifically, we consider that each job i has an amount of work $w_i = p_i$ and it is executed at a constant speed $s_i = 1$. Based on this idea, we extend the existing $\mathcal{NP}$-hardness reduction by fixing an energy budget, so that all jobs have to be executed at the same speed $s_i = 1$ in order to get a feasible schedule.

Theorem 2 *$S1 \mid r_j \mid L_{max}(E)$ problem is strongly $\mathcal{NP}$-hard.*

Proof We construct an instance of $S1 \mid r_j \mid L_{max}(E)$ from an instance of 3-PARTITION as follows. The instance is depicted in Table 1.

- For each integer a_i, $1 \leq i \leq 3n$, we create a job i with $w_i = a_i$, $r_i = 0$ and $q_i = 0$.
- We introduce $n-1$ gadget jobs, where the gadget job i, $3n+1 \leq i \leq 4n-1$, has $w_i = B, r_i = (2i-6n-1)B$ and $q_i = (8n-2i-1)B$.
- We set $E = (2n-1)B$.

i	w_i	r_i	q_i
1	a_1	0	0
2	a_2	0	0
...	...	...	...
$3n$	a_{3n}	0	0
$3n+1$	B	B	$(2n-3)B$
$3n+2$	B	$3B$	$(2n-5)B$
$3n+3$	B	$5B$	$(2n-7)B$
...	...	...	...
$4n-2$	B	$(2n-5)B$	$3B$
$4n-1$	B	$(2n-3)B$	B

Table 1 An instance of $S1 \mid r_j \mid L_{max}(E)$ reduced from an instance of 3-Partition.

We shall prove that there is a feasible schedule σ with $L_{max} = (2n-1)B$ and total energy consumption $E = (2n-1)B$ if and only if there exists a 3-PARTITION of A.

($\Leftarrow$) For the first direction, assume that $A_1, A_2 \dots, A_n$ is a partition of A, where $\sum_{a_i \in A_k} a_i = B$ for $1 \leq k \leq n$. Then, consider the schedule σ where: (i) each job i corresponding to an integer $a_i \in A_k$, $1 \leq k \leq n$, is scheduled during the time interval $[2(k-1)B, (2k-1)B]$, (ii) each gadget job i, $3n+1 \leq i \leq 4n-1$ is scheduled during the time interval $[(2i-6n-1)B, (2i-6n)B]$, and (iii) all jobs are executed at constant speed $s_i = 1$. The schedule σ (see Figure 2) is feasible and attains maximum lateness equal to $L_{max} = (2n-1)B$. The total energy consumed is $E = \sum_{i=1}^{4n-1} w_i s_i^{\alpha-1} = \sum_{i=1}^{4n-1} w_i = (2n-1)B$.

($\Rightarrow$) For the opposite direction, assume that σ is a feasible schedule with $L_{max} = (2n-1)B$ and total energy consumption $E = (2n-1)B$. In σ, each job i, $1 \leq i \leq 3n$, must have completion time $C_i \leq (2n-1)B$ and each gadget job i, $3n+1 \leq i \leq 4n-1$, must have completion time $C_i \leq (2i-6n)B$, since $L_i \leq (2n-1)B$ for every job i. For notational convenience, let $W = (2n-1)B$ be the sum of works of all jobs. Let also p_i be the execution time of job i, $1 \leq i \leq 4n-1$.

It holds also that the completion time of (the last job of) schedule σ is $C_{max} = (2n-1)B$. To see this, assume for the sake of contradiction that $C_{max} < (2n-1)B$. Then, by the convexity of speed-to-power function, it follows that the total energy consumption in σ will be

$$E(\sigma) = \sum_{i=1}^{4n-1} w_i s_i^{\alpha-1} = \sum_{i=1}^{4n-1} w_i \left(\frac{w_i}{p_i}\right)^{\alpha-1} \geq W\left(\frac{W}{C_{max}}\right)^{\alpha-1} > (2n-1)B$$

which is not possible because the energy budget is exceeded. With a similar argument, it can be shown that there will be no idle time during the interval $[0,(2n-1)B]$ in σ. Moreover, due to the convexity of the speed-to-power function, among the schedules with makespan $C_{max} = (2n-1)B$ which have no idle period during $[0,(2n-1)B]$, only the ones in which all the jobs are executed with speed equal to $s_j = 1$ have energy consumption not greater than $E = (2n-1)$. Clearly, σ must be one of these schedules. Hence, every gadget job i, $3n+1 \leq i \leq 4n-1$, is executed within the whole time interval $[(2i-6n-1)B,(2i-6n)B]$ in σ.

So far we have shown that every gadget job i, $3n+1 \leq i \leq 4n-1$, spans in σ the time interval $[(2i-6n-1)B,(2i-6n)B]$, while the other jobs i, $1 \leq i \leq 3n$, span the time intervals $[2(k-1)B,(2k-1)B]$, $1 \leq k \leq n$. Therefore, during any interval $[2(k-1)B,(2k-1)B]$, $1 \leq k \leq n$, there will be executed a set of jobs with total amount of work B. This execution defines a 3-PARTITION for A.

3.2 The on-line case

Let us now turn our attention to the online version of the $S1 \mid r_j \mid L_{max}(E)$ problem. Bansal et al. [4] gave an adversarial strategy for proving that there is no $O(1)$-competitive algorithm for the problem of minimizing the total flow of a set of unit work jobs on a single speed-scalable processor. This adversarial strategy consists of batches of jobs, $B_1, B_2, \ldots, B_k$, with all the jobs in batch B_i released after the online algorithm has finished all the jobs in B_{i-1}. Following a similar strategy it can be proved that the makespan minimization problem, for a given budget of energy, i.e. the problem $S1 \mid r_j,\ w_j = 1 \mid C_{max}(E)$, does not admit an $O(1)$-competitive algorithm. Note that the makespan minimization is a special case of our lateness problem (with $q_i = 0,\ 1 \leq i \leq n$).

Theorem 3 *There is no $O(1)$-competitive algorithm for the online version of the $S1 \mid r_j \mid C_{max}(E)$ problem, even when jobs have unit works.*

Proof In order to prove the theorem, we assume the existence of a ρ-competitive algorithm $\mathcal{A}$, where $\rho > 1$ is a constant. Then, we reach a contradiction by showing that there is an instance of the problem that cannot be feasibly solved by $\mathcal{A}$.

We consider a set of jobs consisting of batches $B_1, B_2, \ldots, B_\ell$, where the batch B_i, $1 \leq i \leq \ell$, contains $n_i = 2^{i-1}$ unit work jobs which all arrive at the same time; the jobs of the batch B_1 are released at the time $r_1 = 0$ while the jobs of the batch B_i, $1 \leq i \leq \ell$, are released at time r_i. We assume that r_i is large enough so that the algorithm $\mathcal{A}$ has completed the jobs in the batches $B_1, \ldots, B_{i-1}$ by r_i.

We denote by $C^*_{max,k}$, $1 \leq k \leq \ell$, the value of the makespan that the optimal offline algorithm achieves for the instance that consists exactly of the jobs in the batches $B_1, B_2, \ldots, B_k$. The term $C^*_{max,k}$ is upper bounded by the makespan of the schedule in which all the jobs in $B_1, B_2, \ldots, B_k$ are assigned equal speeds such that their energy consumption is equal to the energy budget E and they are executed continuously starting at time r_k. Therefore,

$$C^*_{max,k} \leq r_k + \left(\frac{\left(\sum_{i=1}^k n_i\right)^\alpha}{E}\right)^{\frac{1}{\alpha-1}} \tag{19}$$

As $\mathcal{A}$ is a ρ-competitive algorithm, it must complete all jobs of the batches $B_1, B_2, \ldots, B_k$ not later than $\rho \cdot C^*_{max,k}$ independently of the number of batches that our original instance contains. Otherwise, it wouldn't be ρ-competitive for the instance of the problem that consists only of the batches $B_1, B_2, \ldots, B_k$. Let $C_{max,k}$ be the completion time of the jobs in batches $B_1, B_2, \ldots, B_k$ in $\mathcal{A}$'s schedule. Then, it must be the case that

$$C_{max,k} \leq \rho \cdot C^*_{max,k} \tag{20}$$

Let E_k be the energy consumption of the jobs in batch B_k in $\mathcal{A}$'s schedule. Due to the convexity of the speed-to-power function, we have that

$$E_k \geq \frac{n_k^\alpha}{(C_{max,k} - r_k)^{\alpha-1}} \tag{21}$$

By combining inequalities (19), (20), (21) and the fact that $r_k \leq C^*_{max,k}$, we obtain that

$$E_k \geq \frac{n_k^\alpha}{\left(\sum_{i=1}^k n_i\right)^\alpha} \frac{E}{(2\rho - 1)^{\alpha-1}}$$

Since $n_i = 2^{i-1}$ for $1 \leq i \leq k$, we conclude that

$$E_k \geq \frac{E}{2^\alpha (2\rho - 1)^{\alpha-1}}$$

Thus, if the number of batches ℓ is large enough, i.e. $\ell \to \infty$, the algorithm will run out of energy after having completed $\lceil 2^\alpha (2\rho - 1)^{\alpha-1} \rceil$ batches, so it won't be able to finish the batches that follow.

4 Aggregated variant

In this section, we turn our attention to the aggregated variant of the maximum lateness problem, where our objective is to minimize $L_{max} + \beta E$, for some $\beta > 0$. For this variant, in the online case, we are able to overcome the impossibility of obtaining constant-factor competitive algorithms (Theorem 3). Initially, we consider instances in which the jobs have a common release date and we describe how to obtain an optimal offline algorithm for the aggregated variant by slightly modifying our algorithm and its analysis for the budget variant in Section 2. For instances with arbitrary release dates, we explain why our $\mathcal{NP}$-hardness proof for the budget variant implies that the aggregated variant is also $\mathcal{NP}$-hard. Last, we turn our attention to the online case of the aggregated problem in which the jobs arrive over time and we propose a 2-competitive algorithm which schedules the jobs into batches, by repeatedly applying our optimal offline algorithm for jobs with a common release date.

Common release dates. When all jobs are released at the same time, $S1 \mid\mid L_{max} + \beta E$, we are able to derive a polynomial algorithm, by using Algorithm BUD in the following way: suppose that we are given the energy consumption E^* of an optimal schedule minimizing $L_{max} + \beta E$. Then, in order to construct such an optimal schedule, it suffices to apply the optimal algorithm for the budget variant with an energy budget equal to E^*. This means that the optimal schedule for the aggregated variant is a regular schedule, satisfying the properties of Lemma 1 (with budget E^*). However, in order to construct the optimal schedule minimizing $L_{max} + \beta E$, we need to compute E^*. One approach, which has been already suggested in the literature for the total flow time criterion (see [3,4]), would be to perform a binary search procedure in the interval of all possible energy levels. Here, we describe an alternative approach which resembles to the one we followed for the budget variant.

We first formulate the aggregated variant as a convex program similar to the one for the budget variant. Now, we do not introduce a constraint on the total energy consumption, since it is added in the objective function. By applying the KKT conditions, we obtain almost the same structure (properties) of an optimal solution with one single difference: the energy consumption is not specified by a given budget of energy, but it results from the fact that the speed of the first job should always be equal to a fixed value. Specifically, the Property (vii) of Lemma 1 is replaced by the fact that *"the job executed first runs at speed* $s_1 = \left(\frac{1}{(\alpha-1)\beta}\right)^{\alpha}$ *"*. Therefore, in order to obtain an optimal schedule for the aggregated variant, it suffices to do the following: Run lines (1)-(6) of Algorithm BUD. Let σ be the schedule produced. Find the highest-index critical job, $i,\ i \neq 1$, in σ, such that its corresponding sequence, (k, i), has speed $s_i < s_1$. Modify σ such that all jobs $1, 2, \ldots, k-1$ are executed at speed s_1.

Arbitrary release dates. It is not hard to see that if we are given an optimal algorithm for the aggregated variant, then we can obtain an optimal polynomial algorithm for the budget variant by using binary search on the possible values of β and stopping at a value β^* such that the energy consumption of the schedule minimizing $L_{max} + \beta^* E$ is equal to the energy budget. Since, by Theorem 2, the budget variant is $\mathcal{NP}$-hard to solve, we conclude that the aggregated variant is also $\mathcal{NP}$-hard.

Now, we turn our attention in the online version of the aggregated variant and we derive a 2-competitive online algorithm for the $S1 \mid r_j \mid L_{max} + \beta E$ problem. The algorithm schedules the jobs in a number of phases by repeatedly applying the optimal offline algorithm for the $S1 \mid \mid L_{max} + \beta E$ problem. This approach was introduced in [15]. We denote by $\sigma^*(J, t)$ the optimal schedule of a set of jobs J with a common release date t.

Algorithm ALE.

Let J_0 be the set of jobs that arrive at time $t_0 = 0$. In phase 0, jobs in J_0 are scheduled according to $\sigma^*(J_0, 0)$. Let t_1 be the time at which the last job of J_0 is finished, i.e., the end of phase 0, and J_1 be the set of jobs released during $(t_0, t_1]$. In phase 1, jobs in J_1 are scheduled as in $\sigma^*(J_1, t_1)$ and so on. In general, if t_i is the end of phase $i-1$, we denote J_i to be the set of jobs released during $(t_{i-1}, t_i]$. Jobs in J_i are scheduled by computing $\sigma^*(J_i, t_i)$. Next, we analyze the competitive ratio of the algorithm.

Theorem 4 *Algorithm ALE is 2-competitive for the online version of the* $S1 \mid r_j \mid L_{max} + \beta E$ *problem.*

Proof Assume that Algorithm ALE produces a schedule in $\ell + 1$ phases. Recall that the jobs of the i-th phase, i.e., the jobs in J_i, are released during $(t_{i-1}, t_i]$ and scheduled as in $\sigma^*(J_i, t_i)$. Let $L_{max,i} + \beta E_i$ be the cost of $\sigma^*(J_i, t_i)$, where $L_{max,i}$ is the maximum lateness among the jobs in J_i and E_i be the energy consumed by the jobs of J_i. The objective value of the algorithm's schedule is

$$SOL = \max_{0 \leq i \leq \ell}\{L_{max,i}\} + \beta \sum_{i=0}^{\ell} E_i \tag{32}$$

Now, we consider the optimal schedule. To lower bound the objective value OPT of an optimal schedule, we round down the release dates of the jobs; the release dates of the jobs in phase i, are rounded down to t_{i-1}. Let σ_d^* and OPT_d be the optimal schedule for the rounded instance and its cost, respectively. Clearly, any feasible schedule for the initial instance is also feasible for the rounded one. Thus, $OPT \geq OPT_d$.

To lower bound OPT_d we consider a restricted speed-scaling scheduling problem, i.e., a problem where each job can only be executed by a subset of the available processors. The instance of this problem consists of $\ell + 1$ available speed-scalable processors $M_0, M_1, \ldots, M_\ell$ and the set of jobs J, with their release dates rounded down, as before. Jobs in J_0 can only be assigned to the processor M_0 and every job in J_i can only be executed by one of the processors M_0 or $M_i,\ 1 \leq i \leq \ell$. Moreover, it is required that all jobs in $J_i,\ 0 \leq i \leq \ell$, are executed by the same processor. Let

σ_m^*, OPT_m be the optimal schedule and its cost, respectively, for this restricted problem. Obviously, $OPT_d \geq OPT_m$ since σ_d^* is feasible for the restricted scheduling problem.

Let us now describe an optimal schedule σ_m^*. Through a simple exchange argument, it can be shown that the jobs of $J_i,\ 0 \leq i \leq \ell$, in an optimal schedule, are executed by the processor M_i. Moreover, jobs in J_i, for $1 \leq i \leq \ell$, are scheduled according to $\sigma^*(J_i, t_{i-1})$, while for $i = 0$, according to $\sigma^*(J_0, t_0)$. Assume that the maximum lateness of the above schedule, is attained by a job of the set $J_k,\ 0 \leq k \leq \ell$, in the processor M_k. So, let $L_{max}^* = L_{max,k}^*$, where L_{max}^*, $L_{max,k}^*$ is the maximum lateness of the schedules $\sigma_m^*, \sigma^*(J_i, t_{i-1})$, respectively. Let E_i^* be the energy consumption of schedule $\sigma^*(J_i, t_{i-1})$. Then,

$$OPT_m = L_{max,k}^* + \beta \sum_{i=0}^{\ell} E_i^* \tag{33}$$

By considering the schedules $\sigma^*(J_i, t_{i-1})$ and $\sigma^*(J_i, t_i)$, it can be easily shown that $L_{max,i}^* = L_{max,i} - (t_i - t_{i-1})$ and $E_i^* = E_i$. Then, by (32) and (33) it yields that $OPT_m = SOL - (t_k - t_{k-1})$. Note that $t_k - t_{k-1}$ is the total processing time of the jobs in J_{k-1}, in the schedule produced by ALE, which is equal to the processing time of the jobs in J_{k-1} in σ_m^*. Recall also that the last job of each set J_i attains $L_{max,i}$. Thus, $t_k - t_{k-1} \leq L_{max,k-1}^* \leq OPT$. Therefore, $SOL \leq 2OPT$ and Algorithm ALE is 2-competitive for the $S1 \mid r_j \mid L_{max} + \beta E$ problem.

5 Concluding Remarks

We presented positive and negative results for the offline and online power-aware versions of the classical maximum lateness scheduling problem. These results, along with the existing literature on power-aware versions of other problems, like makespan and total flow time, form a strong evidence for the complexity of the power-aware scheduling problems: they are in the same complexity class (polynomial or $\mathcal{NP}$-hard) as their classical versions. For polynomial algorithms, the most promising approach consists of deducing strong structural properties of optimal schedules by applying the KKT conditions on a convex programming formulation of the problem. For $\mathcal{NP}$-hardness results, existing $\mathcal{NP}$-completeness reductions of the corresponding classical problems can be adapted, by forcing all jobs to be executed with speed equal to one and considering the processing times as works. An interesting direction for future work concerns the use of resource (energy) augmentation (see [9,14]) for the online case of the budget variant of the problem, in order to overcome the fact that there is no $O(1)$-competitive deterministic algorithm (see Theorem 3). It is also interesting to improve the competitive ratio of Algorithm ALE, in Section 4, for the aggregated variant.

References

1. S. Albers. Energy-efficient algorithms. *Communications of ACM*, 53(5):86–6, 2010.
2. S. Albers. Algorithms for dynamic speed scaling. In *Symposium on Theoretical Aspects of Computer Science*, volume 9 of *LIPIcs*, pages 1–11. Schloss Dagstuhl, 2011.
3. S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4):article 49, 2007.
4. N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.
5. Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.
6. D.P. Bunde. Power-aware scheduling for makespan and flow. *Journal of Scheduling*, 12(5):489–500, 2009.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.
8. J. R. Jackson. Scheduling a production line to minimize maximum tardiness. Res. Rep. 43, Management Science Research Project, UCLA, 1955.
9. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.
10. E.L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: algorithms and complexity. In *Handbooks in Operations Research and Management Science*, volume 4, pages 445–522. North Holland, 1976.

11. J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.
12. A. Nemirovski, I. Nesterov, and Y. Nesterov. *Interior Point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics, 1994.
13. K. Pruhs, P. Uthaisombut, and G.J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3):article 38, 2008.
14. K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems*, 43(1):67–80, 2008.
15. David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM J. Comput.*, 24(6):1313–1331, 1995.
16. F. F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Symposium on Foundations of Computer Science*, pages 374–382, 1995.