Throughput Maximization in Multiprocessor Speed-Scaling

Eric Angel* Evripidis Bampis † Vincent Chau* Nguyen Kim Thang*

April 15, 2014

Abstract

We are given a set of n jobs that have to be executed on a set of m speed-scalable machines that can vary their speeds dynamically using the energy model introduced in [Yao et al., FOCS'95]. Every job j is characterized by its release date r_j, its deadline d_j, its processing volume $p_{i,j}$ if j is executed on machine i and its weight w_j. We are also given a budget of energy E and our objective is to maximize the weighted throughput, i.e. the total weight of jobs that are completed between their respective release dates and deadlines. We propose a polynomial-time approximation algorithm where the preemption of the jobs is allowed but not their migration. Our algorithm uses a primal-dual approach on a linearized version of a convex program with linear constraints. Furthermore, we present two optimal algorithms for the non-preemptive case where the number of machines is bounded by a fixed constant. More specifically, we consider: *(a)* the case of identical processing volumes, i.e. $p_{i,j} = p$ for every i and j, for which we present a polynomial-time algorithm for the unweighted version, which becomes a pseudopolynomial-time algorithm for the weighted throughput version, and *(b)* the case of agreeable instances, i.e. for which $r_i \leq r_j$ if and only if $d_i \leq d_j$, for which we present a pseudopolynomial-time algorithm. Both algorithms are based on a discretization of the problem and the use of dynamic programming.

1 Introduction

Power management has become a major issue in our days. One of the mechanisms used for saving energy in computing systems is speed-scaling where the speed of the machines can dynamically change over time. We adopt the model first introduced by Yao et al. [26] and we study the multiprocessor scheduling problem of maximizing the throughput of jobs for a given budget of energy. Maximizing throughput, i.e. the number of jobs or the total weight of jobs executed on time for a given budget of energy is a very natural objective in this setting. Indeed mobile devices, such as mobile phones or computers, have a limited energy capacity depending on the quality of their battery, and throughput is one of the most popular objectives in scheduling literature for evaluating the performance of scheduling algorithms for problems involving jobs that are subject to release dates and deadlines [14, 24, 13]. Different variants of the throughput maximization problem in the online speed-scaling setting have been studied in the literature [17, 25, 12, 18]. However, in the off-line context, only recently, an optimal pseudopolynomial-time algorithm has been proposed for the *preemptive*[1] single-machine case [4]. Up to our knowledge no results are known for the throughput maximization problem in the multiprocessor case. In this paper, we address this issue. More specifically, we first consider the case of a set of unrelated machines and we propose a polynomial-time constant-approximation algorithm for

*IBISC, Université d'Evry Val d'Essonne, France.
†Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France.
[1]The execution of a job may be interrupted and resumed later.

the problem of maximizing the weighted throughput in the *preemptive non-migratory*[2] setting. Our algorithm is based on the primal-dual scheme and it is inspired by the approach used in [20] for the online matching problem. In the second part of the paper, we propose exact algorithms for a fixed number of identical parallel machines for instances where the processing volumes of the jobs are all equal, or agreeable instances. Much attention has been paid to these types of instances in the speed-scaling literature (witness for instance [3]). Our algorithms, in this part, are for the non-preemptive case and they are based on a discretization of the problem and the use of dynamic programming.

Problem Definition and Notations In the first part of the paper, we consider the problem for a set of unrelated parallel machines. Formally, there are m unrelated machines and n jobs. Each job j has its release date r_{ij}, deadline d_{ij}, and its processing volume p_{ij} on machine i. Moreover, job j has weight w_j which represents its value. If a job is executed on machine i then it must be entirely processed during its available time interval $[r_{ij}, d_{ij}]$ on that machine without migration. The *weighted throughput* of a schedule is $\sum_{i,j} w_j$ where the sum is taken over jobs j completed on machine i . At any time, a machine can choose a speed to process a job. If the speed of machine i at time t is $s_i(t)$ then the energy power at t is $P_i(s_i(t))$ where P_i is a given convex power energy function of machine i. Typically, one has $P_i(z) := z^{\alpha_i}$ where $2 \leq \alpha_i \leq 3$. The *consumed energy* on machine i is $\int_0^\infty P_i(s_i(t))dt$. Our objective is to maximize the weighted throughput for a given budget of energy E. Hence, the scheduler has to decide the set of jobs which will be executed, assign the jobs to machines and choose appropriate speeds to schedule such jobs without exceeding the energy budget.

In the second part of the paper we consider identical parallel machines where the job parameters (i.e., its release date, deadline, processing volume) are not machine-dependent and the energy power functions are the same for every machine, i.e., $P(z) = z^\alpha$. We study the problem on two families of instances: *(a)* instances with identical processing volumes, i.e. $p_{i,j} = p$ for every i and j, and *(b)* agreeable instances, i.e. for which $r_i \leq r_j$ if and only if $d_i \leq d_j$.

In the sequel, we need the following definition: Given arbitrary convex functions P_i as the energy power functions, define $\Gamma := \max_i \max_{z>0} zP_i'(z)/P_i(z)$. As said before, for the most studied case in the literature one has $P_i(z) = z^{\alpha_i}$, and therefore $\Gamma = \max_i \alpha_i$.

1.1 Our approach and contributions

In this paper, we propose an approximation algorithm for the preemptive non-migratory weighted throughput problem on a set of unrelated speed-scalable machines in Section 2. Instead of studying the problem directly, we study the related problem of minimizing the consumed energy under the constraint that the total weighted throughput must be at least some given throughput demand W.

For the problem of minimizing the energy's consumption under throughput constraint, we present a polynomial time algorithm which has the following property: the consumed energy of the algorithm given a throughput demand W is at most that of an optimal schedule with throughput demand $2(\Gamma+1)W$. The algorithm is based on a primal-dual scheme for mathematical programs with linear constraints and a convex objective function. Specifically, our approach consists in considering a relaxation with convex objective and linear constraints. Then, we linearize the convex objective function and construct a dual program. Using this procedure, the strong duality is not necessarily ensured but the weak duality always holds and that is indeed

[2]This means that the execution of a job may be interrupted and resumed later, but only on the same machine on which it has been started.

the property that we need for our approximation algorithm. The linearization and the dual construction follow the scheme introduced in [20] for online matching. In the relaxation, we also make use of the knapsack inequalities, presented in [16], in order to reduce the integrality gap in the multiprocessor environments. The algorithm follows the standard primal-dual framework: at any time, some dual variables are greedily increased until some dual constraint becomes tight. Then some job is selected and is dispatched to the corresponding machine revealed by the dual constraint. Typically, one will bound the primal objective value by the dual one. In the analysis, instead of comparing the primal and its dual, we bound the primal by the dual of the same relaxation but with the new demand which is $2(\Gamma + 1)$ times larger. An advantage in the analysis is that the feasible solutions of the dual program corresponding to the primal with demand W is also feasible for the dual corresponding to the primal with demand $2(\Gamma + 1)W$.

For the problem of maximizing the throughput under a given budget of energy, we apply a dichotomy search using as subroutine the algorithm for the problem of minimizing the energy's consumption for a given weighted throughput demand. Our algorithm is a $2(\Gamma + 1)(1 + \epsilon)$-approximation for the weighted throughput where $\epsilon > 0$ is an arbitrarily small constant. The algorithm's running time is polynomial in the input size of the problem and $1/\epsilon$. Clearly, one may be interested in finding a tradeoff between the precision and the running time of the algorithm.

In Section 3, we propose exact algorithms for the non-preemptive scheduling on a fixed number of speed-scalable identical machines. By identical machines, we mean that $p_{i,j} = p_j$, i.e. the processing volume of every job is independent of the machine on which it will be executed. Moreover, $r_{ij} = r_j$ and $d_{ij} = d_j$ for every job j and every machine i. We show that for the special case of the problem in which there is a single machine and the release dates and deadlines of the jobs are *agreeable* (for every jobs j and j', if $r_j < r_{j'}$ then $d_j \leq d_{j'}$) the weighted throughput problem is already weakly $\mathcal{NP}$-hard when all the processing volumes are equal. We consider the following two cases (1) jobs have the same processing volume but have arbitrary release dates and deadlines; and (2) jobs have arbitrary processing volumes, but their release dates and deadlines are agreeable. We present pseudo-polynomial time algorithms based on dynamic programming for these variants. Specifically, when all jobs have the same processing volume, our algorithm has running time $O(n^{12m+7}W^2)$ where $W = \sum_j w_j$. Note that when jobs have unit weight, the algorithm has polynomial running time. When jobs are agreeable, our algorithm has running time $O(n^{2m+2}V^{2m+1}Wm)$ where $V = \sum_j p_j$. Using standard techniques, these algorithms may lead to approximation schemes.

1.2 Related work

A series of papers appeared for some online variants of throughput maximization: the first work that considered throughput maximization and speed scaling in the online setting has been presented by Chan et al. [17]. They considered the single machine case with release dates and deadlines and they assumed that there is an upper bound on the machine's speed. They are interested in maximizing the throughput, and minimizing the energy among all the schedules of maximum throughput. They presented an algorithm which is $O(1)$-competitive with respect to both objectives. Li [25] has also considered the maximum throughput when there is an upper bound in the machine's speed and he proposed a 3-approximation greedy algorithm for the throughput and a constant approximation ratio for the energy consumption. In [12], Bansal et al. improved the results of [17], while in [23], Lam et al. studied the 2-machines environment. In [19], Chan et al. defined the energy efficiency of a schedule to be the total amount of work completed in time divided by the total energy usage. Given an efficiency threshold, they considered the problem of finding a schedule of maximum throughput. They showed that no

deterministic algorithm can have competitive ratio less than the ratio of the maximum to the minimum jobs' processing volume. However, by decreasing the energy efficiency of the online algorithm the competitive ratio of the problem becomes constant. Finally, in [18], Chan et al. studied the problem of minimizing the energy plus a rejection penalty. The rejection penalty is a cost incurred for each job which is not completed on time and each job is associated with a value which is its importance. The authors proposed an $O(1)$-competitive algorithm for the case where the speed is unbounded and they showed that no $O(1)$-competitive algorithm exists for the case where the speed is bounded. In what follows, we focus on the offline case. Angel et al. [5] were the first to consider the throughput maximization problem in this setting. They provided a polynomial time algorithm to solve optimally the single-machine problem for agreeable instances. More recently in [4], they proved that there is a pseudo-polynomial time algorithm for solving optimally the preemptive single-machine problem with arbitrary release dates and deadlines and arbitrary processing volume. For the weighted version, the problem is $\mathcal{NP}$-hard even for instances in which all the jobs have common release dates and deadlines. Angel et al. [5] showed that the problem admits a pseudo-polynomial time algorithm for agreeable instances. Furthermore, Antoniadis et al. [8] considered a generalization of the classical knapsack problem where the objective is to maximize the total profit of the chosen items minus the cost incurred by their total weight. The case where the cost functions are convex can be translated in terms of a weighted throughput problem where the objective is to select the most profitable set of jobs taking into account the energy costs. Antoniadis et al. presented a FPTAS and a fast 2-approximation algorithm for the non-preemptive problem where the jobs have no release dates or deadlines.

Up to the best of our knowledge, no works are known for the offline throughput maximization problem in the case of multiple machines. However, many papers consider the closely related problem of minimizing the consumed energy.

For the preemptive single-machine case, Yao et al.[26] in their seminal paper proposed an optimal polynomial-time algorithm. Since then, a lot of papers appears in the literature (see [1]). Antoniadis and Huang [7] have considered the non-preemptive energy minimization problem. They proved that the non-preemptive single-machine case is strongly NP-hard even for laminar instances [3] and they proposed a $2^{5\alpha-4}$-approximation algorithm. This result has been improved recently in [10] where the authors proposed a $2^{\alpha-1}(1+\varepsilon)\tilde{B}_\alpha$-approximation algorithm, where $\tilde{B}_\alpha$ is the generalized Bell number. For instances in which all the jobs have the same processing volume, Bampis et al. [9] gave a 2^α-approximation for the single-machine case. However the complexity status of this problem remained open. In this paper, we settle this question even for the identical machine case where the number of the machine is a fixed constant. Notice that independently, Huang et al. in [22] proposed a polynomial-time algorithm for the single machine case.

The multiple machine case where the preemption and the migration of jobs are allowed can be solved in polynomial time in [2], [6] and [11]. Albers et al. [3] considered the multiple machine problem where the preemption of jobs is allowed but not their migration. They showed that the problem is polynomial-time solvable for agreeable instances when the jobs have the same processing volumes. They have also showed that it becomes strongly NP-hard for general instances even for jobs with equal processing volumes and for this case they proposed an $(\alpha^\alpha 2^{4\alpha})$-approximation algorithm. For the case where the jobs have arbitrary processing volumes, they showed that the problem is NP-hard even for instances with common release dates and common deadlines. Albers et al. proposed a $2(2-1/m)^\alpha$-approximation algorithm for instances with common release dates, or common deadlines, and an $(\alpha^\alpha 2^{4\alpha})$-approximation algorithm for

[3] In a laminar instance for any pair of jobs J_i and J_j, either $[r_j, d_j] \subseteq [r_i, d_i]$, $[r_i, d_i] \subseteq [r_j, d_j]$, or $[r_i, d_i] \cap [r_j, d_j] = \emptyset$.

instances with agreeable deadlines. Greiner et al. [21] proposed a B_α-approximation algorithm for general instances, where B_α is the α-th Bell number. Recently, the approximation ratio for agreeable instances has been improved to $(2-1/m)^{\alpha-1}$ in [9]. For the non-preemptive multiple machine energy minimization problem, the only known result is a non-constant approximation algorithm presented in [9].

2 Approximation Algorithms for Preemptive Scheduling

In Section 2.1, we first study a related problem in which we look for an algorithm that minimizes the consumed energy under the constraint of throughput demand. Then in Section 2.2 we use that algorithm as a sub-routine to derive an algorithm for the problem of maximizing throughput under the energy constraint.

2.1 Energy Minimization with Throughput Demand Constraint

In the problem, there are n jobs and m unrelated machines. A job j has release date r_{ij}, deadline d_{ij}, weight w_j and processing volume p_{ij} if it is scheduled on machine i. Given throughput demand W, the scheduler needs to choose a subset of jobs, assign them to the machines and decide the speed to process these job in such a way that the total weight (throughput) of completed jobs is at least W and the consumed energy is minimized. Jobs are allowed to be processed preemptively but without migration.

Let x_{ij}'s be variables indicating whether job j is scheduled in machine i. Let $s_{ij}(t)$'s be the variable representing the speed that the machine i processes job j at time t. The problem can be formulated as the following primal convex relaxation ($\mathcal{P}$).

$$\begin{aligned}
\min \quad & \sum_i \int_0^\infty P_i(s_i(t))dt && && (\mathcal{P})\\
\text{subject to} \quad & s_i(t) = \sum_j s_{ij}(t) && \forall i,t &\\
& \sum_i x_{ij} \le 1 && \forall j && (1)\\
& \int_{r_{ij}}^{d_{ij}} s_{ij}(t)dt \ge p_{ij}x_{ij} && \forall i,j && (2)\\
& \sum_i \sum_{j:j\notin S} w_j^S x_{ij} \ge W - w(S) && \forall S \subset \{1,\dots,n\} && (3)\\
& x_{ij}, s_{ij}(t) \ge 0 && \forall i,j,t &
\end{aligned}$$

In the relaxation, constraints (1) ensures that a job can be chosen at most once. Constraints (2) guarantee that job j must be completed if it is assigned to machine i. To satisfy the throughput demand constraint, we use the knapsack inequalities (3) introduced in [16]. Note that in the constraints, S is a subset of jobs, $w(S) = \sum_{j\in S} w_j$ and $w_j^S := \min\{w_j, W - w(S)\}$. Intuitively, if S is the set of jobs which will be completed then one need to cover $W - w(S)$ amount of throughput over jobs not in S in order to satisfy the demand. Those constraints reduce significantly the integrality gap of the relaxation compared to the natural constraint $\sum_{ij} w_j x_{ij} \ge W$.

Define functions $Q_i(z) := P_i(z) - zP_i'(z)$ for every machine i. Consider the following a dual

program $(\mathcal{D})$.

$$\max \quad \sum_S (W - w(S))\beta_S + \sum_i \int_0^\infty Q_i(v_i(t))dt - \sum_j \gamma_j \qquad (\mathcal{D})$$

$$\text{s.t} \quad \lambda_{ij} \leq P_i'(v_i(t)) \qquad \forall i, j, \forall t \in [r_{ij}, d_{ij}] \qquad (4)$$

$$\sum_{S: j \notin S} w_j^S \beta_S \leq \gamma_j + \lambda_{ij} p_{ij} \qquad \forall i, j \qquad (5)$$

$$\lambda_{ij}, \gamma_j, v_i(t) \geq 0 \qquad \forall i, j, \forall t$$

The construction of the dual $(\mathcal{D})$ is inspired by [20] and is obtained by linearizing the convex objective of the primal. By that procedure the strong duality is not necessarily guaranteed but the weak duality always holds. Indeed we only need the weak duality for approximation algorithms. In fact, the dual $(\mathcal{D})$ gives a meaningful lower bound that we will exploit to design our approximation algorithm.

Lemma 1 (Weak Duality)**.** *The optimal value of the dual program $(\mathcal{D})$ is at most the optimal value of the primal program $(\mathcal{P})$.*

Proof. As P_i is convex, for every t and functions s_i and v_i, we have

$$\begin{aligned} P_i(s_i(t)) &\geq P_i(v_i(t)) + (s_i(t) - v_i(t))P_i'(v_i(t)) \\ &= P_i'(v_i(t))s_i(t) + Q_i(v_i(t)) \end{aligned} \qquad (6)$$

Notice that if $v_i(t)$ is fixed then $P_i(s_i(t))$ has a lower bound in linear form (since in that case $P_i'(v_i(t))$ and $Q(v_i(t))$ are constants). We use that lower bound to derive the dual. Fix functions $v_i(t)$ for every $1 \leq i \leq m$. Consider the following linear program and its dual in the usual sense of linear programming.

By strong LP duality, the optimal value of theses primal and dual programs are equal. Denote that value with $OPT(v_1, \ldots, v_m)$.

Let $O_{\mathcal{P}}$ be the optimal value of the primal program $(\mathcal{P})$. Hence, for every choice of $v_i(t)$, we have a lower bound on $O_{\mathcal{P}}$, i.e., $O_{\mathcal{P}} \geq OPT(v_1, \ldots, v_m) + \sum_i \int_0^\infty Q_i(v_i(t))$ by (6). So $O_{\mathcal{P}} \geq \max_{v_1,\ldots,v_m} OPT(v_1, \ldots, v_m) + \sum_i \int_0^\infty Q_i(v_i(t))$ where $v_i(t)$'s are feasible solutions for $(\mathcal{D})$. The latter is the optimal value of the dual program $(\mathcal{D})$. Hence, the lemma follows. □

The primal/dual programs $(\mathcal{P})$ and $(\mathcal{D})$ highlights main ideas for the algorithm. Intuitively, if a job j is assigned to machine i then one must increase the speed of job j in machine i at $\arg\min P_i'(v_i(t))$ in order to always satisfy the constraint (4). Moreover, when constraint (5) becomes tight for some job j and machine i, one could assign j to i in order to continue to raise some β_S and increase the dual objective. The formal algorithm is given as follows.

$$
\begin{aligned}
\min \quad & \sum_i \int_0^\infty P_i'(v_i(t)) \sum_j s_{ij}(t) dt & \\
& \sum_i x_{ij} \le 1 & \forall j \\
& \int_{r_{ij}}^{d_{ij}} s_{ij}(t) dt \ge p_{ij} x_{ij} & \forall i, j \\
& \sum_i \sum_{j: j \notin S} w_j^S x_{ij} \ge W - w(S) & \forall S \\
& x_{ij}, s_{ij}(t) \ge 0 & \forall i, j, t
\end{aligned}
$$

$$
\begin{aligned}
\max \quad & \sum_S (W - w(S)) \beta_S - \sum_j \gamma_j & \\
& \lambda_{ij} \le P_i'(v_i(t)) & \forall i, j, \forall t \in [r_{ij}, d_{ij}] \\
& \sum_{S: j \notin S} w_j^S \beta_S \le \gamma_j + \lambda_{ij} p_{ij} & \forall i, j \\
& \lambda_{ij}, \gamma_j, v_i(t) \ge 0 & \forall i, j, \forall t
\end{aligned}
$$

Figure 1: Strong duality for LP

Algorithm 1 Minimizing the consumed energy under the throughput constraint

1: Initially, set $s_i(t), s_{ij}(t), v_i(t)$ and λ_{ij}, γ_j equal to 0 for every job j, machine i and time t.
2: Initially, $\mathcal{T} \leftarrow \emptyset$.
3: **while** $W > w(\mathcal{T})$ **do**
4: **for** every job $j \notin \mathcal{T}$ and every machine i **do**
5: Continuously increase $s_{ij}(t)$ at $\arg\min P_i'(v_i(t))$ for $r_{ij} \le t \le d_{ij}$ and simultaneously update $v_i(t) \leftarrow v_i(t) + s_{ij}(t)$ until $\int_{r_{ij}}^{d_{ij}} s_{ij} dt = p_{ij}$.
6: Set $\lambda_{ij} \leftarrow \min_{r_{ij} \le t \le d_{ij}} P_i'(v_i(t))$.
7: Reset $v_i(t)$ as before, i.e., $v_i(t) \leftarrow v_i(t) - s_{ij}(t)$ for every $t \in [r_{ij}, d_{ij}]$.
8: **end for**
9: Continuously increase $\beta_{\mathcal{T}}$ until $\sum_{S: j \notin S} w_j^S \beta_S = p_{ij} \lambda_{ij}$ for some job j and machine i.
10: Assign job j to machine i. Set $s_i(t) \leftarrow s_i(t) + s_{ij}(t)$ and $v_i(t) \leftarrow s_i(t)$ for every t.
11: Set $\mathcal{T} \leftarrow \mathcal{T} \cup \{j\}$. Moreover, set $\gamma_j \leftarrow p_{ij} \lambda_{ij}$.
12: Reset $\lambda_{i'j} \leftarrow 0$ and $s_{i'j}(t) \leftarrow 0$ for every $i' \neq i$.
13: **end while**

In the algorithm $\arg\min P_i'(v_i(t))$ for $r_{ij} \le t \le d_{ij}$ is defined as $\{t : t \in [r_{ij}, d_{ij}]$ and $P_i'(v_i(t)) = \min_{r_{ij} \le x \le d_{ij}} P_i'(v_i(x))\}$, this is usually a set of intervals, and thus the speed s_{ij} is increased simultaneously on a set of intervals. Notice also that since P_i is a convex function, P_i' is non decreasing. Hence, in line 5 of the algorithm, $\arg\min P_i'(v_i(t))$ can be replaced by $\arg\min v_i(t)$; so we can avoid the computation of the derivative $P_i'(z)$. Note that in the end of the algorithm variables $v_i(t)$ is indeed equal to $s_i(t)$ — the speed of machine i for every i. Given the assignment of jobs and the speed function $s_i(t)$ of each machine i returned by the algorithm, one can process jobs on each machine by the earliest deadline first order.

The algorithm is illustrated by an example given in the appendix.

Lemma 2. *The solution β_S, γ_j and $v_i(t)$ for every i, j, S, t constructed by Algorithm 1 is feasible for the dual $(\mathcal{D})$.*

Proof. By the algorithm, variables λ_{ij}'s and variables $v_i(t)$'s are maintained in such a way that the constraints (4) are always satisfied. Moreover, by the construction of variables β_S's, λ_{ij}'s and γ_j's, the constraints (5) are ensured (for every machine and every job). □

Theorem 1. *The consumed energy of the schedule returned by the algorithm with a throughput demand of W is at most the energy of the optimal schedule with a throughput demand $2(\Gamma+1)W$.*

Proof. Let $OPT(2(\Gamma+1)W)$ be the energy consumed by the optimal schedule with the throughput demand $2(\Gamma+1)W$. By Lemma 1, we have that

$$OPT(2(\Gamma+1)W) \geq \sum_S (2(\Gamma+1)W - w(S))\beta_S + \sum_i \int_0^\infty Q(v_i(t))dt - \sum_j \gamma_j$$

where the variables β_S, v_i, γ_j satisfy the same constraints in the dual $(\mathcal{D})$. Therefore, it is sufficient to prove that latter quantity is larger than the consumed energy of the schedule returned by the algorithm with the throughput demand W, denoted by $ALG(W)$. Specifically, we will prove a stronger claim. For β_S, γ_j and v_i (which is equal to s_i) in the feasible dual solution constructed by Algorithm 1 with the throughput demand W, it always holds that

$$2(\Gamma+1)\sum_S (W - w(S))\beta_S + \sum_i \int_0^\infty Q(s_i(t))dt - \sum_j \gamma_j \geq \sum_i \int_0^\infty P_i(s_i(t))dt.$$

By the algorithm, we have that

$$\sum_{i,j} p_{ij}\lambda_{ij} = \sum_{i,j:j\in\mathcal{T}} p_{ij}\lambda_{ij} = \sum_{j\in\mathcal{T}}\sum_{S:j\notin S} w_j^S \beta_S = \sum_S \beta_S \Bigg(\sum_{j\notin S, j\in\mathcal{T}} w_j^S\Bigg) \leq 2\sum_S \beta_S(W - w(S)) \quad (7)$$

By the algorithm in the first sum $\sum_{i,j} p_{ij}\lambda_{ij}$, each term $p_{ij}\lambda_{ij} \neq 0$ iff $j \in \mathcal{T}$ and j is assigned to i. In the third sum, $\beta_S \neq 0$ iff S equals $\mathcal{T}$ at some step during the execution of the algorithm. Thus, we consider only such sets in that sum. Let j^* be the last element added to $\mathcal{T}$. For $S \subset \mathcal{T}\setminus\{j^*\}$ and $\beta_S > 0$, by the while loop condition $w(S) + \sum_{j\notin S, j\in\mathcal{T}\setminus\{j^*\}} w_j^S < W$. Moreover, $w_{j^*}^S \leq w_{j^*}^{\mathcal{T}} \leq W - w(\mathcal{T}) \leq W - w(S)$. Hence, $\sum_{j\notin S, j\in\mathcal{T}} w_j^S \leq 2(W - w(S))$ and the inequality (7) follows.

Fix a machine i and let $\{1, \ldots, k\}$ be the set of jobs assigned to machine i (renaming jobs if necessary). Let $u_{i1}(t), \ldots, u_{ik}(t)$ be the speed of machine i at time t after assigning jobs $1, \ldots, k$, respectively. In other words, $u_{i\ell}(t) = \sum_{j=1}^{\ell} s_{ij}(t)$ for every $1 \leq \ell \leq k$. By the algorithm, we have $\lambda_{i\ell} = \min_{r_\ell \leq t \leq d_\ell} P_i'(u_{i\ell}(t))$ for every $1 \leq \ell \leq k$. As every job ℓ is completed in machine i,

$\int_{r_\ell}^{d_\ell} s_{i\ell}(t)dt = p_{i\ell}$. Note that $s_{i\ell}(t) > 0$ only at t in $\arg\min_{r_\ell \le t \le d_\ell} P_i'(u_{i\ell}(t))$. Thus,

$$\begin{aligned}
\sum_{\ell=1}^{k} \lambda_{i\ell} p_{i\ell} &= \sum_{\ell=1}^{k} \int_{r_\ell}^{d_\ell} s_{i\ell}(t) P_i'\Big(\sum_{j=1}^{\ell} s_{ij}(t)\Big) dt \\
&= \sum_{\ell=1}^{k} \int_{0}^{\infty} s_{i\ell}(t) P_i'\Big(\sum_{j=1}^{\ell} s_{ij}(t)\Big) dt \\
&\ge \sum_{\ell=1}^{k} \int_{0}^{\infty} \Big[P_i\Big(\sum_{j=1}^{\ell} s_{ij}(t)\Big) - P_i\Big(\sum_{j=1}^{\ell-1} s_{ij}(t)\Big)\Big] dt \\
&= \int_{0}^{\infty} \Big[P_i(u_{ik}(t)) - P_i(0)\Big] dt \\
&= \int_{0}^{\infty} P_i(s_i(t)) dt \qquad (8)
\end{aligned}$$

where in the second equality, note that $s_{i\ell}(t) = 0$ for $t \notin [r_\ell, d_\ell]$; the inequality is due to the convexity of P_i.

As inequality (8) holds for every machines i, summing over all machines we get

$$\sum_{i,j} p_{ij}\lambda_{ij} \ge \sum_i \int_0^\infty P_i(s_i(t))dt.$$

Together with (7), we deduce that

$$\begin{aligned}
&2(\Gamma+1)\sum_S \beta_S(W - w(S)) + \sum_i \int_0^\infty Q(s_i(t))dt - \sum_j \gamma_j \\
&\ge \sum_{i,j} p_{ij}\lambda_{ij} + \Gamma \sum_i \int_0^\infty P_i(s_i(t))dt + \sum_i \int_0^\infty Q(s_i(t))dt - \sum_j \gamma_j \\
&\ge \sum_i \int_0^\infty P_i(s_i(t))dt = ALG(W).
\end{aligned}$$

where the last inequality is due to the definition of Γ (recall that $\Gamma = \max_i \max_z zP_i'(z)/P_i(z)$ for every z such that $P(z) > 0$) and $\gamma_j = \sum_i \lambda_{ij} p_{ij}$ for every job j (by the algorithm). □

Corollary 1. *If the demand $W = \min_j w_j$ then the energy induced by Algorithm 1 is optimal (compared to the optimal solution with the same demand).*

Proof. If the demand $W = \min_j w_j$ then it is sufficient to complete any job. By Algorithm 1, it is optimal if one needs only one job completed. □

Corollary 2. *For single machine setting, the consumed energy of the schedule returned by the algorithm with a throughput demand of W is at most that of the optimal schedule with a throughput demand $2\Gamma \cdot W$.*

Proof. For single machine setting, we can consider a relaxation similar to $(\mathcal{P})$ without constraints (1) and without machine index i for all variables. The dual construction, the algorithm and the analysis remain the same. Observe that now there is no dual variable γ_j. By that point we can improve the factor from $2(\Gamma + 1)$ to 2Γ. □

Note that a special case of the single machine setting is the minimum knapsack problem. In the latter, we are given a set of n items, item j has size p_j and value w_j. Moreover, given a demand W, the goal is to find a subset of items having minimum total size such that the total value is at least W. The problem corresponds to the single machine setting where all jobs have the same span, i.e., $[r_{ij}, d_{ij}] = [r, d]$ for all jobs $j \neq j'$; item size and value correspond to job processing volume and weight, respectively; and the energy power $P(z) = z$. Carnes and Shmoys [15] gave a 2-approximation primal-dual algorithm for the minimum knapsack problem. That result is a special case of Corollary 2 where $\Gamma = 1$ for linear function $P(z)$.

2.2 Throughput Maximization with Energy Constraint

We use the algorithm in the previous section as a sub-routine and make a dichotomy search in the feasible domain of the total throughput. The formal algorithm is given as follows.

Algorithm 2 Maximizing throughput under the energy constraint

For a throughput demand W, denote $E(W)$ the consumed energy due to Algorithm 1.
Let $\epsilon > 0$ be a constant.
Initially, set $W \leftarrow \min_j w_j$ and $\overline{W} \leftarrow \sum_j w_j$ where the sum is taken over all jobs j.
if $E(W) > E$ **then**
  return the total throughput is 0
end if
while $E((1+\epsilon)W) \leq E$ and $(1+\epsilon)W \leq \overline{W}$ **do**
  $W \leftarrow (1+\epsilon)W$.
end while
return the schedule which is the solution of Algorithm 1 with throughput demand W.

Theorem 2. *Given an energy budget E, Algorithm 2 is $2(\Gamma+1)(1+\epsilon)$-approximation in throughput for arbitrarily small $\epsilon > 0$. The running time of the algorithm is polynomial in the size of input and $1/\epsilon$.*

Proof. Let W^* be the optimal throughput with the energy budget E. Let W be the throughput returned by Algorithm 2. If the energy budget is not enough for Algorithm 1 to complete any job, meaning that $E(\min_j w_j) > E$, then by Corollary 1 optimal solution cannot complete any job neither. The theorem trivially holds. Now assume that $W > 0$.

We argue the theorem by contradiction, suppose that $W^* > 2(\Gamma+1)(1+\epsilon)W$. By Theorem 1, the consumed energy of the optimal schedule must be at least $E((1+\epsilon)W)$. By the while loop condition of the algorithm, either $(1+\epsilon)W > \overline{W}$ or $E((1+\epsilon)W) > E$. If the former holds then $W^* > \overline{W}$, the total weight of jobs in the instance (contradiction). If the latter holds then the optimal solution violates the energy budget. Therefore, we deduce that $W^* \leq 2(\Gamma+1)(1+\epsilon)W$.

In Algorithm 2, the number of iterations in the while loop is proportional to the size of the input and $1/\epsilon$. As Algorithm 1 is polynomial, the running time of Algorithm 2 is polynomial in the size of input and $1/\epsilon$. □

3 Exact Algorithms for Non-Preemptive Scheduling

3.1 Preliminaries

Notations In this section, we consider schedules without preemption with a fixed number m of identical machines. So the parameters of a job j are the same on every machine (so the

machine index is dropped out). Without loss of generality, we assume that all parameters of the problem such as release dates, deadlines and processing volumes of jobs are *integer*. We rename jobs in non-decreasing order of their deadlines, i.e. $d_1 \leq d_2 \leq \ldots \leq d_n$. We denote by $r_{\min} := \min_{1 \leq j \leq n} r_j$ the minimum release date. Define Ω as the set of release dates and deadlines (EDF), i.e., $\Omega := \{r_j | j = 1, \ldots, n\} \cup \{d_j | j = 1, \ldots, n\}$. Let $J(k, a, b) := \{j | j \leq k \text{ and } a \leq r_j < b\}$ be the set of jobs among the k first ones w.r.t. the EDF order, whose release dates are within a and b. We consider *time vectors* $\mathbf{a} = (a_1, a_2, \ldots, a_m) \in \mathbb{R}_+^m$ where each component a_i is a time associated to the machines i for $1 \leq i \leq m$. We say that $\mathbf{a} \preceq \mathbf{b}$ if $a_i \leq b_i$ for every $1 \leq i \leq m$. Moreover, $\mathbf{a} \prec \mathbf{b}$ if $\mathbf{a} \preceq \mathbf{b}$ and $\mathbf{a} \neq \mathbf{b}$. The relation $\preceq$ is a partial order over the time vectors. Given a vector $\mathbf{a}$, we denote by $a_{\min} := \min_{1 \leq i \leq m} a_i$.

Observations We give some simple observations on non-preemptive scheduling with the objective of maximizing throughput under the energy constraint. First, it is well known that due to the convexity of the power function $P(z) := z^\alpha$, each job runs at a constant speed during its whole execution in an optimal schedule. This follows from Jensen's Inequality. Second, for a restricted version of the problem in which there is a single machine, jobs have the same processing volume and are agreeable, the problem is already $\mathcal{NP}$-hard. That is proved by a simple reduction from KNAPSACK.

Proposition 1. *The problem of maximizing the weighted throughput on the case where jobs have agreeable deadline and have the same processing volume is weakly $\mathcal{NP}$-hard.*

Proof. Let Π be the the weighted throughput problem on the case where jobs have agreeable deadline and have the same processing volume. In an instance of the KNAPSACK problem we are given a set of n items, each item j has a value κ_j and a size c_j. Given a capacity C and a value K, we are asked for a subset of items with total value at least K and total size at most C.

Given an instance of the KNAPSACK problem, construct an instance of problem Π as follows. For each item j, create a job j with $r_j := \sum_{\ell=1}^{j-1} c_\ell$, $d_j := \sum_{\ell=1}^{j} c_\ell = r_j + c_j$, $w_j := \kappa_j$ and $p_j := 1$. Moreover, we set $E := C$, i.e. the budget of energy is equal to C.

We claim that the instance of the KNAPSACK problem is feasible if and only if there is a feasible schedule for problem Π of total weighted throughput at least K.

Assume that the instance of the KNAPSACK is feasible. Therefore, there exists a subset of items J' such that $\sum_{j \in J'} \kappa_j \geq K$ and $\sum_{j \in J'} c_i \leq C$. Then we can schedule all jobs corresponding to item in J' with constant speed equal to 1. That gives a feasible schedule with total energy consumption at most C and the total weight at least K.

For the opposite direction of our claim, assume there is a feasible schedule for problem Π of total weighted throughput at least K. Let J' be the jobs which are completed on time in this schedule. Clearly, due to the convexity of the speed-to-power function, the schedule that executes the jobs in J' with constant speed is also feasible. Since the latter schedule is feasible, we have that $\sum_{j \in J'} (d_j - r_j) \leq C$. Moreover, $\sum_{j \in J'} w_j \geq K$. Therefore, the items which correspond to the jobs in J' form a feasible solution for the KNAPSACK. □

The hardness result rules out the possibility of polynomial-time exact algorithms for the problem. However, as the problem is weakly $\mathcal{NP}$-hard, there is still possibility for approximation schemes. In the following sections, we show pseudo-polynomial-time exact algorithms for instances with equal processing volume jobs and agreeable jobs.

3.2 Equal Processing Volume

In this section, we assume that $p_j = p$ for every job j.

Definition 1. *Let* $\Theta_{a,b} := \{a + \ell \cdot \frac{b-a}{k} \mid k = 1, \ldots, n \text{ and } \ell = 0, \ldots, k \text{ and } a \leq b\}$. *Moreover,* $\Theta := \bigcup\{\Theta_{a,b} | a, b \in \Omega\}$.

The following lemma gives an observation on the structure of an optimal schedule.

Lemma 3. *There exists an optimal schedule in which the starting time and completion time of each job belong to the set* Θ.

Proof. Let $\mathcal{O}$ be an optimal schedule and $\mathcal{O}_i$ be the corresponding schedule $\mathcal{O}$ on machine i. $\mathcal{O}_i$ can be partitioned into successive blocks of jobs where the blocks are separated by idle-time periods. Consider a block B and decompose B into maximal sub-blocks $B_1, \ldots, B_k$ such that all the jobs executed inside a sub-block B_ℓ are scheduled with the same common speed s_ℓ for $1 \leq \ell \leq k$. Let j and j' be two consecutive jobs such that j and j' belong to two consecutive sub-blocks, let's say B_ℓ and $B_{\ell+1}$. Then either $s_\ell > s_{\ell+1}$ or $s_\ell < s_{\ell+1}$. In the first case, the completion time of job j (which is also the starting time of job j') is necessarily d_j, otherwise we could obtain a better schedule by decreasing (resp. increasing) the speed of job j (resp. j'). For the second case, a similar argument shows that the completion time of job j is necessarily $r_{j'}$. Hence, each sub-block begins and finishes at a date which belong to Ω.

Consider a sub-block B_ℓ and let a, b be its starting and completion times. As jobs have the same volume and the jobs scheduled in B_ℓ are processed non-preemptively by the same speed, their starting and completion times must belong to $\Theta_{a,b}$. □

Using Lemma 3 we can assume that each job is processed at some speed which belong to the following set.

Definition 2. *Let* $\Lambda := \{\frac{\ell \cdot p}{b-a} \mid \ell = 1, \ldots, n \text{ and } a, b \in \Omega \text{ and } a < b\}$ *be the set of different speeds.*

Definition 3. *For* $0 \leq w \leq W$, *define* $E_k(\mathbf{a}, \mathbf{b}, w, e)$ *as the minimum energy consumption of a non-preemptive (non-migration) schedule* $\mathcal{S}$ *such that*

- $S \subset J(k, a_{\min}, b_{\min})$ *and* $\sum_{j \in S} w_j \geq w$ *where* S *is the set of jobs scheduled in* $\mathcal{S}$,
- *if* $j \in S$ *is assigned to machine* i *then it is entirely processed in* $[a_i, b_i]$ *for every* $1 \leq i \leq m$,
- $\mathbf{a} \preceq \mathbf{b}$,
- *for some machine* $1 \leq h \leq m$, *it is idle during interval* $[a_h, e]$,
- *for arbitrary machines* $1 \leq i \neq i' \leq m$, $b_{i'}$ *is at least the last starting time of a job in machine* i.

Note that $E_k(\mathbf{a}, \mathbf{b}, w, e) = \infty$ if no such schedule $\mathcal{S}$ exists.

Proposition 2. *One has*

$$E_0(\mathbf{a}, \mathbf{b}, 0, e) = 0$$
$$E_0(\mathbf{a}, \mathbf{b}, w, e) = +\infty \ \forall w \neq 0$$
$$E_k(\mathbf{a}, \mathbf{b}, w, e) = \min \begin{cases} E_{k-1}(\mathbf{a}, \mathbf{b}, w, e) \\ E' \end{cases}$$

where

$$E' = \min_{\substack{\mathbf{u}\in\Theta^m \\ \mathbf{a}\prec\mathbf{u}\prec\mathbf{b} \\ s\in\Lambda, 1\leq h\leq m, \\ e'=u_h+\frac{p}{s} \\ r_k\leq u_h<e'\leq d_k \\ 0\leq w'\leq w-w_k}} \left\{ \begin{array}{c} E_{k-1}(\mathbf{a},\mathbf{u},w',e) \\ +\frac{p^\alpha}{(e'-u_h)^{\alpha-1}} \\ +E_{k-1}(\mathbf{u},\mathbf{b},w-w'-w_k,e') \end{array} \right\}$$

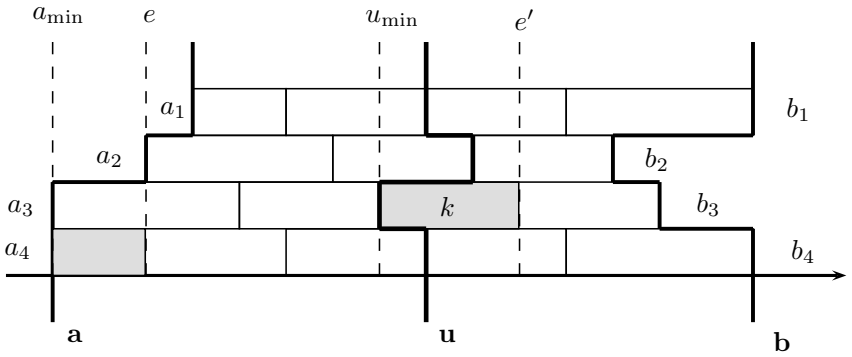

Figure 2: Illustration of Proposition 2

Proof. The base case for E_0 is straightforward. We will prove the recursive formula for $E_k(\mathbf{a},\mathbf{b},w,e)$. There are two cases: (1) either in the schedule that realizes $E_k(\mathbf{a},\mathbf{b},w,e)$, job k is not chosen, so $E_k(\mathbf{a},\mathbf{b},w,e) = E_{k-1}(\mathbf{a},\mathbf{b},w,e)$; (2) or k is chosen in that schedule. In the following, we are interested by that second case.

We first prove that $E_k(\mathbf{a},\mathbf{b},w,e) \leq E'$ Fix some arbitrary time vector $\mathbf{a} \prec \mathbf{u} \prec \mathbf{b}$ and weight $0 < w' < w - w_k$ and time e' such that $r_k \leq e' = u_i + \frac{p}{s} \leq d_k$ for some $s \in \Lambda$ and some machine h. Consider a schedule $\mathcal{S}_1$ that realizes $E_{k-1}(\mathbf{a},\mathbf{u},w',e)$ and $\mathcal{S}_2$ a schedule that realizes $E_{k-1}(\mathbf{u},\mathbf{b},w-w'-w_k,e')$. We build a schedule with $\mathcal{S}_1$ from $\mathbf{a}$ to $\mathbf{u}$ and with $\mathcal{S}_2$ from $\mathbf{u}$ to $\mathbf{b}$ and job k scheduled within $\mathcal{S}_2$ during $[u_i, e']$ on machine h. Recall that by definition of $E_{k-1}(\mathbf{u},\mathbf{b},w-w'-w_k,e')$, machine h does not execute any job during $[u_h, e']$. Obviously, the subsets $J(k-1, a_{\min}, u_{\min})$ and $J(k, u_{\min}, b_{\min})$ do not intersect, so this is a feasible schedule which costs at most

$$E_{k-1}(\mathbf{a},\mathbf{u},w',e) + \frac{p^\alpha}{(e'-u_h)^{\alpha-1}} + E_{k-1}(\mathbf{u},\mathbf{b},w-w'-w_k,e').$$

As that holds for every time vector $\mathbf{a} \prec \mathbf{u} \prec \mathbf{b}$ and weight $0 < w' < w - w_k$ and time e' such that $r_k \leq e' = u_h + \frac{p}{s} \leq d_k$ for some $s \in \Lambda$ and some machine h, we deduce that $E_k(\mathbf{a},\mathbf{b},w,e) \leq E'$.

We now prove that $E' \leq E_k(\mathbf{a},\mathbf{b},w,e)$ Let $\mathcal{S}$ be the schedule that realizes $E_k(\mathbf{a},\mathbf{b},w,e)$ in which the starting time of job k is maximal. Suppose that job k is scheduled on machine h and its starting time is denoted as u_h. For every machine $i \neq h$, define $u_i \geq u_h$ be the earliest completion time of a job which is completed after u_h on machine i by schedule $\mathcal{S}$. If no job is completed after u_h on machine i then define $u_i = b_i$. Hence, we have a time vector $\mathbf{a} \prec \mathbf{u} = (u_1, \ldots, u_m) \prec \mathbf{b}$.

We split $\mathcal{S}$ into two sub-schedules $\mathcal{S}_1 \subseteq \mathcal{S}$ and $\mathcal{S}_2 = \mathcal{S} \setminus (\mathcal{S}_1 \cup \{k\})$ such that $j \in \mathcal{S}_1$ if it is started and completed in $[a_i, u_i]$ for some machine i. Note that such job j has release date $r_j \in [a_{\min}, u_{\min}[$.

We claim that for every job $j \in \mathcal{S}_2$, $r_j \geq u_h$ where $u_h = u_{\min}$ by the definition of vector $\mathbf{u}$. By contradiction, suppose that some job $j \in \mathcal{S}_2$ has $r_j \leq u_h$, meaning that job j is available at the starting time of job k. By definition of $\mathcal{S}_1$ and $\mathcal{S}_2$, job j is started after the starting time of job k. Moreover, $j < k$ means that $d_j \leq d_k$. Thus, we can swap jobs j and k (without modifying the machine speeds). Since all jobs have the same volume, this operation is feasible. The new schedule has the same energy cost while the starting time of job k is strictly larger. That contradicts the definition of $\mathcal{S}$.

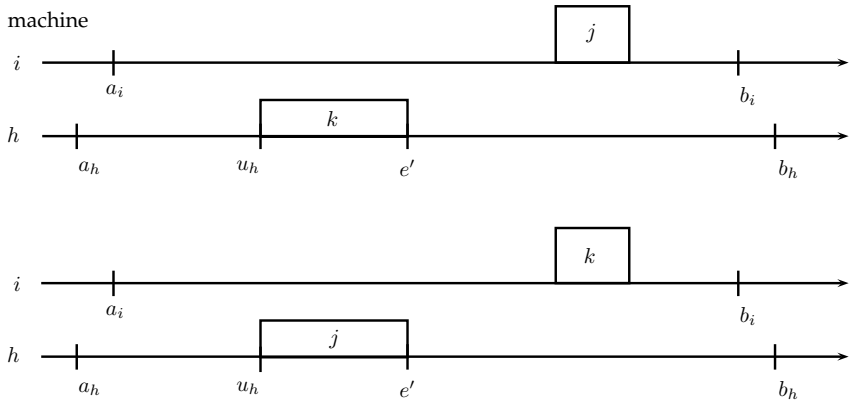

Figure 3: Illustration of the swap argument

Therefore, all jobs in $\mathcal{S}_1$ have release dates in $[a_{\min}, u_{\min}[$ and all jobs in $\mathcal{S}_2$ have release dates in $[u_{\min}, b_{\min}[$. Moreover, with the definition of vector $\mathbf{u}$, the schedules $\mathcal{S}_1$ and $\mathcal{S}_2$ are valid (according to Definition 3). Let $s_{hk} \in \Lambda$ be the speed that machine h processes job k in $\mathcal{S}$. Hence, the consumed energies by schedules $\mathcal{S}_1$ and $\mathcal{S}_2$ are at least $E_{k-1}(\mathbf{a}, \mathbf{u}, w', e)$ and $E_{k-1}(\mathbf{u}, \mathbf{b}, w - w_k - w', u_h + \frac{p}{s_{hk}})$ where w' is the total weight of jobs in $\mathcal{S}_1$. We have

$$E_k(\mathbf{a}, \mathbf{b}, w, e) \geq E_{k-1}(\mathbf{a}, \mathbf{u}, w', e) + \frac{p^\alpha}{(u_h + \frac{p}{s_{hk}} - u_h)^{\alpha-1}} + E_{k-1}(\mathbf{u}, \mathbf{b}, w - w_k - w', u_h + \frac{p}{s_{hk}}) = E'.$$

Therefore, we deduce that $E' = E_k(\mathbf{a}, \mathbf{b}, w, e)$ in case job k is chosen in the schedule that realizes $E_k(\mathbf{a}, \mathbf{b}, w, e)$. The proposition follows. □

Theorem 3. *The dynamic program in Proposition 2 has a running time of $O(n^{12m+7}W^2)$.*

Proof. Denote $r_{\min} = \min_{1 \leq j \leq n} r_j$. Given an energy budget E, the objective function is $\max\{w \mid E_n((r_{\min}, \ldots, r_{\min}), (d_n, \ldots, d_n), w, r_{\min}) \leq E\}$. The values of $E_k(\mathbf{a}, \mathbf{b}, w, e)$ are stored in a multi-dimensional array of size $O(|\Theta|^{2m}|\Lambda|nW)$. Each value need $O(|\Lambda||\Theta|^m\ mW)$ time to be computed thanks to Proposition 2. Thus we have a total running time of $O(|\Theta|^{3m}|\Lambda|^2 nmW^2)$. This leads to an overall time complexity $O(n^{12m+7}mW^2)$. □

3.3 Agreeable Jobs

In this section, we focus on another important family of instances. More precisely, we assume that the jobs have *agreeable* deadlines, i.e. for any pair of jobs i and j, one has $r_i \leq r_j$ if and only if $d_i \leq d_j$.

Based on Definition 1, we can extend the set of starting and completion times for each job into the set Φ.

Definition 4. *Let* $\Phi_{a,b} := \{a + \ell \cdot \frac{b-a}{k} \mid k = 1, \ldots, V \text{ and } \ell = 0, \ldots, k\}$ *with* $V := \sum_j p_j$, *and* $\Phi := \bigcup\{\Phi_{a,b} \mid a, b \in \Omega\}$.

The following lemmas show the structure of an optimal schedule that we will use in order to design our algorithm.

Lemma 4. *There exists an optimal solution in which all jobs in each machine are scheduled according to the Earliest Deadline First (*EDF*) order without preemption.*

Proof. Let $\mathcal{O}$ be an optimal schedule. Let j and j' be two consecutive jobs that are scheduled on the same machine i in $\mathcal{O}$. We suppose that job j is scheduled before job j' with $d_{j'} \leq d_j$. Let a (resp. b) be the starting time (resp. completion time) of job j' (resp. job j) in $\mathcal{O}$. Then, we have necessarily $r_{j'} \leq r_j \leq a < b \leq d_{j'} \leq d_j$. The execution of jobs j and j' can be swapped in the time interval $[a, b]$. Thus we obtain a feasible schedule $\mathcal{O}'$ in which job j' is scheduled before job j with the same energy consumption. □

Lemma 5. *There exists an optimal* EDF *schedule* $\mathcal{O}$ *in which each job in* $\mathcal{O}$ *has its starting time and its completion time that belong to the set* Φ.

Proof. We proceed as in Lemma 3. We partition an optimal schedule $\mathcal{O}$ into blocks and sub-blocks where the starting and completion times of every sub-blocks belong to the set Λ. Consider an arbitrary sub-block. As all the parameters are integer, the total volume of the sub-block is also an integer in $[0, V]$ and the total number of jobs processed in the sub-block is bounded by the total volume. Thus the starting and completion times of any job in the sub-block must belong to the set Φ. □

By Lemma 5, we can assume that each job is processed with a speed that belongs to the following set.

Definition 5. *Let* $\Delta := \{\frac{i}{b-a} \mid i = 1, \ldots, V \text{ and } a, b \in \Omega\}$ *be the set of different speeds.*

Definition 6. *For* $1 \leq w \leq W$, *define* $F_k(\mathbf{b}, w)$ *as the minimum energy consumption of an non-preemptive (and a non-migratory) schedule* $\mathcal{S}$ *such that:*

- $S \subseteq J(k, r_{\min}, b_{\min})$ *and* $\sum_{j \in S} w_j \geq w$ *where* S *is the set of jobs scheduled in* $\mathcal{S}$
- *if* $j \in S$ *is assigned to machine* i *then it is entirely processed in* $[r_{\min}, b_i]$ *for every* $1 \leq i \leq m$.

Note that $F_k(\mathbf{b}, w) = \infty$ if no such schedule $\mathcal{S}$ exists.

For a vector $\mathbf{b}$ and a speed $s \in \Delta$, let $\texttt{prec}_k(\mathbf{b}, s)$ be the set of vectors $\mathbf{a} \prec \mathbf{b}$ such that there always exists some machine $1 \leq h \leq m$ with the following properties:

$$\begin{cases} r_k \leq a_h = \min\{b_h, d_k\} - \dfrac{p_k}{s}, \ a_h \in \Phi \\ a_i = b_i \ \forall i \neq h. \end{cases}$$

Proposition 3. *One has*

$$\begin{aligned} F_0(\mathbf{b}, 0) &= 0 \\ F_0(\mathbf{b}, w) &= +\infty \ \forall w \neq 0 \\ F_k(\mathbf{b}, w) &= \min\{F_{k-1}(\mathbf{b}, w), F'\} \end{aligned}$$

where

$$F' = \min_{\substack{s\in\Delta \\ \mathbf{a}=prec_k(\mathbf{b},s)}} \left\{F_{k-1}(\mathbf{a}, w - w_k) + p_k s^{\alpha-1}\right\}$$

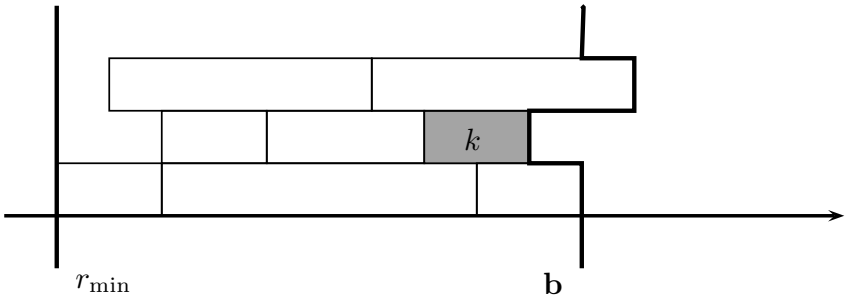

Figure 4: Illustration of Proposition 3

Proof. The base case for F_0 is straightforward. We will prove the recursive formula for $F_k(\mathbf{b}, w)$. There are two cases: (1) either in the schedule that realizes $F_k(\mathbf{b}, w)$, job k is not chosen, so $F_k(\mathbf{b}, w) = F_{k-1}(\mathbf{b}, w)$; (2) or k is chosen in that schedule. In the following, we are interested in the case when k is chosen.

We first prove that $F_k(\mathbf{b}, w) \leq F'$ Fix some arbitrary time vector $\mathbf{a} \prec \mathbf{b}$ and a_i such that $r_k \leq a_i = \min\{b_i, d_k\} - \frac{p_k}{s}$ for some speed $s \in \Delta$ and some machine i. Then we have $\mathbf{a} = (b_1, \ldots, \min\{b_i, d_k\} - \frac{p_k}{s}, \ldots, b_m)$. Consider a schedule $\mathcal{S}$ that realizes $F_{k-1}(\mathbf{a}, w - w_k)$. We build a schedule with $\mathcal{S}$ from $(r_{min}, \ldots, r_{min})$ to $\mathbf{a}$ and job k is scheduled on machine i during $[a_i, \min\{b_i, d_k\}]$ and an idle period during $[\min\{b_i, d_k\}, b_i]$. So this is a feasible schedule which costs at most

$$F_{k-1}(\mathbf{a}, w - w_k) + p_k s^{\alpha-1}$$

As that holds for every time vector $\mathbf{a} \prec \mathbf{b}$ and some speed $s \in \Delta$ and some machine i, we deduce that $F_k(\mathbf{b}, w) \leq F'$.

We now prove that $F' \leq F_k(\mathbf{b}, w)$ Let $\mathcal{S}$ be the schedule that realizes $F_k(\mathbf{b}, w)$ in which the starting time of job k is maximal. We consider the sub-schedule $\mathcal{S}' = \mathcal{S} \setminus \{k\}$. We claim that all the jobs of $\mathcal{S}'$ are completed before $\mathbf{a} \in \mathtt{prec}_k(\mathbf{b}, s)$ which is the vector obtained from $\mathbf{b}$ after removing job k.

Hence the cost of the schedule $\mathcal{S}'$ is at least $F_{k-1}(\mathbf{a}, w - w_k)$. Thus,

$$F_k(\mathbf{b}, w) \geq F_{k-1}(\mathbf{a}, w - w_k) + p_k(s)^{\alpha-1} = F'$$

Therefore, we deduce that $F' = F_k(\mathbf{b}, w)$ in case job k is chosen in the schedule that realizes $F_k(\mathbf{b}, w)$. The proposition follows. □

Theorem 4. *The dynamic programming in Proposition 3 has a total running time of* $O(n^{2m+2}V^{2m+1}Wm)$.

Proof. Given an energy budget E, the objective function is $\max\{w \mid F_n(\mathbf{b}, w) \leq E,\ 1 \leq w \leq W,\ \mathbf{b} \in \Phi^m :\ d_1 \leq b_i \leq d_n\ \forall i\}$. The values of $F_k(\mathbf{b}, w)$ are stored in a multi-dimensional array of size $O(n|\Theta|^m W)$. Each value need $O(|\Delta|Wm)$ time to be computed thanks to Proposition 3. Thus we have a total running time of $O(n|\Theta|^m|\Delta|Wm)$. This leads to an overall time complexity $O(n^{2m+2}V^{2m+1}\ Wm)$. □

References

[1] S. Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.

[2] S. Albers, A. Antoniadis, and G. Greiner. On multi-processor speed scaling with migration: extended abstract. In *Proc. 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 279–288. ACM, 2011.

[3] S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *Proc. 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 289–298. ACM, 2007.

[4] E. Angel, E. Bampis, and V. Chau. Throughput maximization in the speed-scaling setting. *to appear in STACS*, 2014.

[5] E. Angel, E. Bampis, V. Chau, and D. Letsios. Throughput maximization for speed-scaling with agreeable deadlines. In *Proc. 10th International Conference Theory and Applications of Models of Computation (TAMC)*, volume 7876 of *LNCS*, pages 10–19. Springer, 2013.

[6] E. Angel, E. Bampis, F. Kacem, and D. Letsios. Speed scaling on parallel processors with migration. In *Proc. 18th International Conference Euro-Par*, volume 7484 of *LNCS*, pages 128–140. Springer, 2012.

[7] A. Antoniadis and C.-C. Huang. Non-preemptive speed scaling. *J. Scheduling*, 16(4):385–394, 2013.

[8] A. Antoniadis, C.-C. Huang, S. Ott, and J. Verschae. How to pack your items when you have to buy your knapsack. In *MFCS*, volume 8087 of *LNCS*, pages 62–73. Springer, 2013.

[9] E. Bampis, A. Kononov, D. Letsios, G. Lucarelli, and I. Nemparis. From preemptive to non-preemptive speed-scaling scheduling. In *Proc. 19th International Conference, Computing and Combinatorics (COCOON)*, pages 134–146, 2013.

[10] E. Bampis, A. Kononov, D. Letsios, G. Lucarelli, and M. Sviridenko. Energy efficient scheduling and routing via randomized rounding. In *FSTTCS*, 2013.

[11] E. Bampis, D. Letsios, and G. Lucarelli. Green scheduling, flows and matchings. In *Proc. 23rd International Symposium on Algorithms and Computation (ISAAC)*, pages 106–115, 2012.

[12] N. Bansal, H.-L. Chan, T. W. Lam, and L.-K. Lee. Scheduling for speed bounded processors. In *ICALP (1)*, volume 5125 of *LNCS*, pages 409–420. Springer, 2008.

[13] P. Baptiste. An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Oper. Res. Lett.*, 24(4):175–180, 1999.

[14] P. Brucker. *Scheduling Algorithms*. Springer Publishing Company, Incorporated, 5th edition, 2010.

[15] T. Carnes and D. B. Shmoys. Primal-dual schema for capacitated covering problems. In *Proc. 13th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 288–302, 2008.

[16] R. D. Carr, L. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 106–115, 2000.

[17] H.-L. Chan, W.-T. Chan, T. W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong. Energy efficient online deadline scheduling. In *SODA*, pages 795–804. SIAM, 2007.

[18] H.-L. Chan, T. W. Lam, and R. Li. Tradeoff between energy and throughput for online deadline scheduling. In *WAOA*, volume 6534 of *LNCS*, pages 59–70. Springer, 2010.

[19] J. W.-T. Chan, T. W. Lam, K.-S. Mak, and P. W. H. Wong. Online deadline scheduling with bounded energy efficiency. In *TAMC*, volume 4484 of *LNCS*, pages 416–427. Springer, 2007.

[20] N. R. Devanur and K. Jain. Online matching with concave returns. In *Proc. 44th ACM Symposium on Theory of Computing*, pages 137–144, 2012.

[21] G. Greiner, T. Nonner, and A. Souza. The bell is ringing in speed-scaled multiprocessor scheduling. In *Proc. 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 11–18. ACM, 2009.

[22] C.-C. Huang and S. Ott. New results for non-preemptive speed scaling. Research report, Max-Planck-Institut für Informatik, 2013.

[23] T. W. Lam, L.-K. Lee, I. K.-K. To, and P. W. H. Wong. Energy efficient deadline scheduling in two processor systems. In *ISAAC*, volume 4835 of *LNCS*, pages 476–487. Springer, 2007.

[24] E. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. volume 26, pages 125–133. Baltzer Science Publishers, Baarn/Kluwer Academic Publishers, 1990.

[25] M. Li. Approximation algorithms for variable voltage processors: Min energy, max throughput and online heuristics. *Theor. Comput. Sci.*, 412(32):4074–4080, 2011.

[26] F. F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382. IEEE Computer Society, 1995.

A Execution of Algorithm 1

In this example, we have $m = 2$ unrelated machines, $n = 4$ jobs and each job have the same weight, i.e. $w_j = 1 \forall j$. We want to compute the energy's consumption when we have to choose $W = 3$ jobs according to our algorithm.

Let $P(z) = z^\alpha$ with $\alpha = 3$ be the power function of the machines. And let the derivative function $P'(z) = 3z^2$.

The processing volume of each job is given in the following table.

$i \backslash j$	1	2	3	4
1	1	3	4	2
2	2	5	3	1

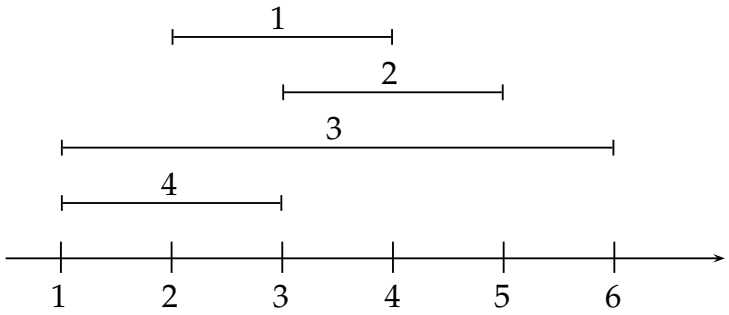

Figure 5: Instance of 4 jobs with the respective release date and deadline

Step 1 At this step, the set of chosen jobs is $T = \emptyset$

We continuously increase the speed $s_{ij}(t)$ for each job j and each machine i with $rj \leq t \leq dj$. Then we obtain the value of $\lambda_{ij} \leftarrow \min_{r_j \leq t \leq d_j} P'(v_i(t))$.

$i \backslash j$	1	2	3	4
1	$P'(\frac{1}{2}) = \frac{3}{4}$	$P'(\frac{3}{2}) = \frac{27}{4}$	$P'(\frac{4}{5}) = \frac{48}{25}$	$P'(1) = 3$
2	$P'(1) = 3$	$P'(\frac{5}{2}) = \frac{75}{4}$	$P'(\frac{3}{5}) = \frac{27}{25}$	$P'(\frac{1}{2}) = \frac{3}{4}$

Table 1: Table of λ_{ij} at Step 1

$i \backslash j$	1	2	3	4
1	3/4	81/4	192/25	6
2	6	625/4	81/25	3/4

Table 2: Table of $\lambda_{ij} p_{ij}$ at Step 1

We continuously increase $\beta_{\mathcal{T}}$ until $\sum_{S: j \notin S} w_j^S \beta_S = p_{ij} \lambda_{ij}$ for some job j and machine i.

Since $\beta_{\mathcal{S}} = 0 \ \forall \mathcal{S}$ at this step and we can only modify the value of $\beta_{\mathcal{T}} = \beta_\emptyset$, then we have to find the maximum value of $\beta_\emptyset$ such that one of the constraint becomes tight.

$w_j^\emptyset \beta_\emptyset = \min\{p_{ij} \lambda_{ij}\} = \frac{3}{4}$ and $\gamma_1 = \frac{3}{4}$

Thus Job 1 is affected to machine 1 and $T = \{1\}$

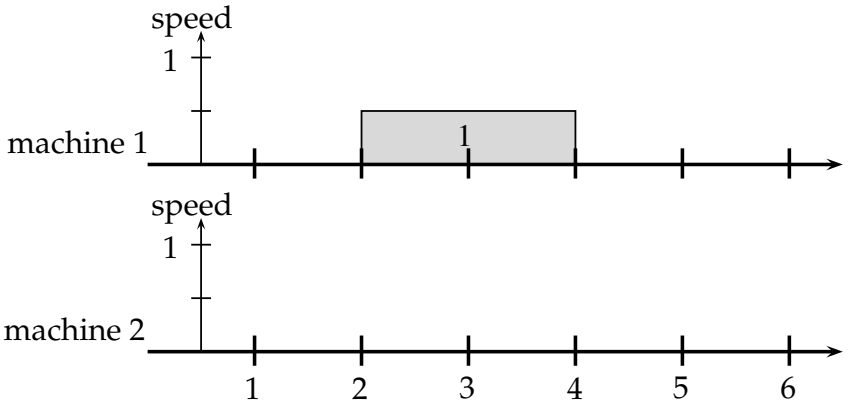

Figure 6: Speed profile $v_i(t)$ at the end of Step 1

Step 2 At this step, the set of chosen jobs is $T = \{1\}$ and the speed profile $v_i(t)$ can be found in Figure 6

$i\backslash j$	2	3	4
1	$P'(\frac{7}{4}) = \frac{147}{16}$	$P'(1) = 3$	$P'(\frac{5}{4}) = \frac{75}{16}$
2	$P'(\frac{5}{2}) = \frac{75}{4}$	$P'(\frac{3}{5}) = \frac{27}{25}$	$P'(\frac{1}{2}) = \frac{3}{4}$

Table 3: Table of λ_{ij} at Step 2

$i\backslash j$	2	3	4
1	441/16	12	150/16
2	625/4	81/25	3/4

Table 4: Table of $\lambda_{ij}p_{ij}$ at Step 2

At this step we have only $\beta_\emptyset$ which is positive. Then we have $\beta_{\{1\}} = 0$

$$\begin{aligned} w_j^\emptyset \beta_\emptyset + w_j^{\{1\}} \beta_{\{1\}} &= \min\{p_{ij}\lambda_{ij}\} \\ \frac{3}{4} + \beta_{\{1\}} &= \min\{p_{ij}\lambda_{ij}\} \\ \beta_{\{1\}} &= 0 \end{aligned}$$

Job 4 is affected to machine 2, $\gamma_4 = \frac{3}{4}$ and $T = \{1, 4\}$.

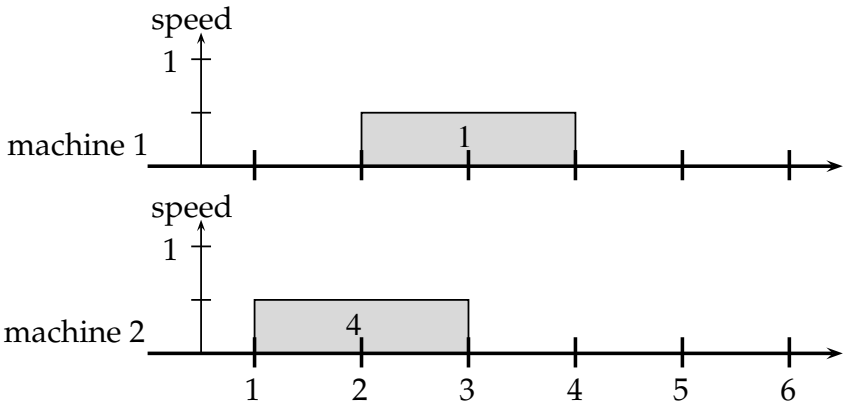

Figure 7: Speed profile $v_i(t)$ at the end of Step 2

Step 3 $T = \{1, 4\}$

$i\backslash j$	2	3
1	$P'(\frac{7}{4}) = \frac{147}{16}$	$P'(1) = 3$
2	$P'(\frac{5}{2}) = \frac{75}{4}$	$P'(\frac{4}{5}) = \frac{48}{25}$

Table 5: Table of λ_{ij} at Step 3

$i\backslash j$	2	3
1	441/16	12
2	625/4	144/25

Table 6: Table of $\lambda_{ij}p_{ij}$ at Step 3

$$\begin{aligned} w_j^{\emptyset}\beta_{\emptyset} + w_j^{\{1\}}\beta_{\{1\}} + w_j^{\{1,4\}}\beta_{\{1,4\}} &= \min\{p_{ij}\lambda_{ij}\} \\ \beta_{\emptyset} + \beta_{\{1\}} + \beta_{\{1,4\}} &= \min\{p_{ij}\lambda_{ij}\} \\ \beta_{\{1,4\}} &= \frac{144}{25} - \frac{3}{4} \\ \beta_{\{1,4\}} &= \frac{501}{100} \end{aligned}$$

Job 3 is affected to machine 2, $\gamma_3 = \frac{144}{25}$ and $T = \{1, 3, 4\}$.

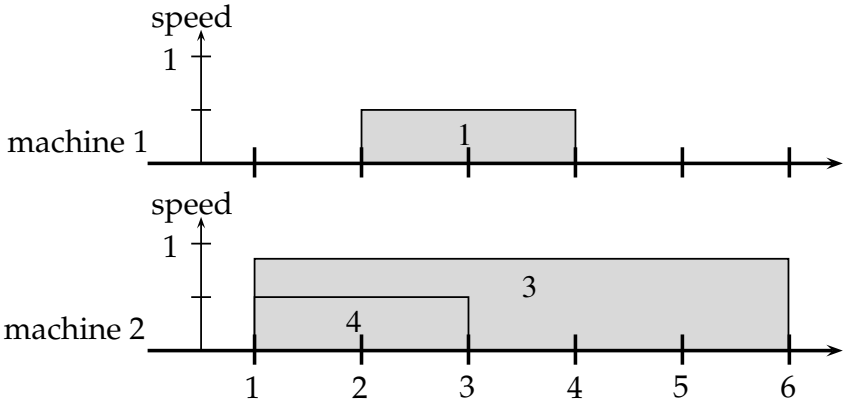

Figure 8: Speed profile $v_i(t)$ at the end of Step 3