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

Throughput Maximization for Speed-Scaling with Agreeable Deadlines

Eric Angel · Evripidis Bampis · Vincent Chau · Dimitrios Letsios

Received: date / Accepted: date

Abstract We study the following energy-efficient scheduling problem. We are given a set of n jobs which have to be scheduled by a single processor whose speed can be varied dynamically. Each job J_j is characterized by a processing requirement (work) p_j, a release date r_j and a deadline d_j. We are also given a budget of energy E which must not be exceeded and our objective is to maximize the throughput (i.e. the number of jobs which are completed on time). We show that the problem can be solved optimally via dynamic programming in $O(n^4 \log n \log P)$ time when all jobs have the same release date, where P is the sum of the processing requirements of the jobs. For the more general case with agreeable deadlines where the jobs can be ordered so that, for every $i < j$, it holds that $r_i \leq r_j$ and $d_i \leq d_j$, we propose an optimal dynamic programming algorithm which runs in $O(n^6 \log n \log P)$

This work has been supported by the ANR project TODO (09-EMER-010), by PHC CAI YUANPEI (27927VE) and by the ALGONOW project of the THALES program.

A preliminary version of this work appeared in the proceedings of the *10th International Conference on Theory and Applications of Models of Computation (TAMC)*, p. 10-19, 2013.

Eric Angel
IBISC, Université d'Evry Val d'Essonne
E-mail: angel@ibisc.fr

Evripidis Bampis
LIP6, Université Pierre et Marie Curie
E-mail: Evripidis.Bampis@lip6.fr

Vincent Chau
Department of Computer Science, City University of Hong Kong
E-mail: vincchau@cityu.edu.hk

Dimitrios Letsios
Lehrstuhl Theoretische Informatik, Fakultät für Informatik, Technische Universität München
E-mail: letsios@in.tum.de

time. In addition, we consider the weighted case where every job J_j is also associated with a weight w_j and we are interested in maximizing the weighted throughput (i.e. the total weight of the jobs which are completed on time). For this case, we show that the problem becomes $\mathcal{NP}$-hard in the ordinary sense even when all jobs have the same release date and we propose a pseudo-polynomial time algorithm for agreeable instances.

Keywords Throughput · Speed-Scaling · Scheduling

1 Introduction

The problem of scheduling n jobs with release dates and deadlines on a single processor that can vary its speed dynamically with the objective of minimizing the energy consumption has been first studied in the seminal paper by Yao, Demers and Shenker [14]. In this paper, we consider the problem of maximizing the throughput without exceeding a given budget of energy.

Formally, we are given a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$, where each job J_j is characterized by a processing requirement (work) p_j, a release date r_j and a deadline d_j. W.l.o.g. we suppose that the earliest released job is released at $t = 0$. We assume that the jobs have to be executed by a single speed-scalable processor, i.e. a processor which can vary its speed over time (at a given time, the processor's speed can be any non-negative value). The processor can execute at most one job at each time. We measure the processor's speed in units of executed work per unit of time. If $s(t)$ is the speed of the processor at time t, then the total amount of work executed by the processor during an interval of time $[t, t')$ is equal to $\int_t^{t'} s(u)du$. Moreover, we assume that the processor's power consumption is a convex function of its speed. Specifically, at any time t, the power consumption of the processor is $P(t) = s(t)^\alpha$, where $\alpha > 1$ is a constant. Since the power is defined as the rate of change of the energy consumption, the total energy consumption of the processor during an interval $[t, t')$ is $\int_t^{t'} s(u)^\alpha du$. Note that if the processor runs at a constant speed s during an interval of time $[t, t')$, then it executes $(t' - t) \cdot s$ units of work and it consumes $(t' - t) \cdot s^\alpha$ units of energy. Each job J_j can start being executed at its release date r_j or after. Moreover, we allow preemptions of jobs, i.e. a job may be executed, suspended and resumed later from the point of suspension. Given a budget of energy E which must not be exceeded, our objective is to find a schedule of maximum throughput, where the throughput of a schedule is defined as the number of jobs which are completed on time, i.e. before their deadline. Observe that a job is completed on time if it is entirely executed during the interval $[r_j, d_j)$. By extending the well-known 3-field notation by Graham et al. [7], this problem can be denoted as $1|pmtn, r_j|\sum U_j(E)$.

We also consider the weighted version of the problem where every job J_j is also associated with a weight w_j and the objective is no more the maximization of the cardinality of the set of jobs that are completed on time, but the maximization of the sum of their weights. We denote this problem as

$1|pmtn, r_j|\sum w_j U_j(E)$. In what follows, we consider instances of the problem in which all jobs have the same release date and another important and more general family of instances, the agreeable instances, in which the jobs can be ordered so that, for every $i < j$, it holds that $r_i \leq r_j$ and $d_i \leq d_j$.

Related Work and Contributions

In classical scheduling, throughput maximization has been studied extensively. In particular, in the problem $1|pmtn, r_j|\sum U_j$ which is a classical scheduling problem, we are given a set of jobs $J = \{J_1, J_2, \ldots, J_n\}$ that have to be executed by a single processor. Each job J_j is associated with a processing time p_j, a release date r_j and a deadline d_j. The objective is to find a schedule of maximum throughput. This problem is polynomially-time solvable and the fastest known algorithm for general instances is in $O(n^4)$ [3]. When all the release dates are equal, i.e. the problem $1||\sum U_j$, it can be solved in $O(n \log n)$ time with Moore's algorithm [12]. Finally, if the jobs have agreeable deadlines, i.e. the problem $1|pmtn, r_j, agreeable|\sum U_j$, then Lawler's algorithm [10] solves the problem optimally in $O(n \log n)$ time. The problem $1|pmtn, r_j|\sum w_j U_j$, where each job is also associated with a weight and the objective is to maximize the weighted throughput, in $\mathcal{NP}$-hard and there exists a pseudo-polynomial time algorithm by Lawler [9].

The current state of the art includes some works on variants of the speed scaling problem with the objective of maximizing the throughput mainly in the online setting. The first work that considered online throughput maximization in speed scaling was by Chan et al. [4]. They considered the online problem of scheduling a set of jobs with release dates and deadlines on a single processor with an upper bound on its speed and their objective was maximizing the throughput while minimizing the energy among all schedules of maximum throughput. For this problem, they presented an algorithm which is $O(1)$-competitive with respect to both objectives. Bansal et al. [2] improved the results in [4] and Lam et al. [8] extended them for multiprocessor environments.

In [6], Chan et al. defined the energy efficiency of a schedule to be the total amount of work completed on time divided by the total energy usage. Given an energy efficiency threshold, they considered the problem of finding a schedule of maximum throughput. They showed that no deterministic algorithm can have competitive ratio better than $O(\frac{p_{max}}{p_{min}})$, i.e. the maximum over the minimum job processing requirement. However, they obtained a constant-factor competitive algorithm with energy efficiency augmentation.

Furthermore, Chan et al. [5] studied the problem of minimizing the energy plus rejection penalties. For a given job, the rejection penalty is the cost incurred if the job is not completed on time. 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.

Finally, there exists also a work on speed scaling with the throughput objective in the offline setting. Li [11] considered the maximum throughput when there is an upper bound on the processor's speed and he proposed a 3-approximation greedy algorithm for the throughput and a constant approximation ratio for the energy consumption.

It has to be noticed that, after the conference version of this paper, Angel et al. [1] proposed an optimal pseudo-polynomial time algorithm solving the general version of the problem $1|pmtn, r_j|\sum w_j U_j(E)$. Moreover, another important metric for the quality of a schedule is flow time. In speed scaling, the study of flow time was initiated by Pruhs et al. [13].

This paper is organized as follows. Initially, we present an optimal algorithm for the case where all jobs have the same release date $r_j = 0$, i.e. for the problem $1||\sum U_j(E)$[1]. Then, we present an optimal algorithm for the more general case with agreeable instances, i.e. $1|r_j, agreeable|\sum U_j(E)$[1]. The reason for presenting both algorithms is that the former algorithm is slightly better than the latter one in terms of worst-case running time. The first algorithm runs in $O(n^4 \log n \log P)$ time while the running time of the second one is $O(n^6 \log n \log P)$. Finally, we consider the even more general case in which the jobs are associated with weights we are interested in maximizing the weighted throughput. We show that the problem $1||\sum w_j U_j(E)$ is $\mathcal{NP}$-hard in the ordinary sense and we propose a pseudo-polynomial time algorithm for $1|r_j, agreeable|\sum w_j U_j(E)$.

2 Initial Remarks

Given that the processor's speed can be varied, a reasonable distinction of the scheduling problems that can be considered is the following:

- **FS** (Fixed Speed): The processor has a fixed speed which implies directly a processing time for each job. In this case, the scheduler has to decide which job must be executed at each time. This is the classical scheduling setting.
- **CS** (Constant Speed): The processor's speed is not known in advance but it can only run at a single speed during the whole time horizon. In this context, the scheduler has to define a single value of speed at which the processor will run and the job executed at each time.
- **SS** (Scalable Speed): The processor's speed can be varied over the time and, at each time, the scheduler has to determine not only which job to run, but the processor's speed as well.

At this point, let us make a remark. Assume that we are provided an optimal algorithm for a FS scheduling problem, i.e. for a classical scheduling problem. Then, we can use this algorithm as a black box and binary search in order to solve the corresponding CS problem. This observation is a key ingredient for designing our algorithms.

[1] There is always an optimal non-preemptive schedule for this problem even if preemptions are allowed (see Property 1).

3 Properties of Optimal Schedules

Among the schedules of maximum throughput, we try to find the one of minimum energy consumption. Therefore, if we knew by an oracle the set of jobs J^*, $J^* \subseteq J$, which are completed on time in an optimal solution, we would simply have to apply an optimal algorithm for $1|pmtn, r_j, d_j|E$ for the jobs in J^* in order to determine a minimum energy schedule of maximum throughput for our problem. Based on this observation, we can use in our analysis some properties of an optimal schedule for $1|pmtn, r_j, d_j|E$.

Let $t_1, t_2, \ldots, t_k$ be the time points which correspond to all the possible release dates and deadlines of the jobs. We number the t_i values in increasing order, i.e. $t_1 < t_2 < \ldots < t_k$. The following theorem can be found in [14].

Theorem 1 *A feasible schedule for $1|pmtn, r_j, d_j|E$ is optimal if and only if all the following hold:*

1. *Each job J_j is executed at a constant speed s_j.*
2. *The processor is not idle at any time t such that $t \in [r_j, d_j)$, for all $J_j \in J$.*
3. *The processor runs at a constant speed during any interval $[t_i, t_{i+1})$, for $1 \leq i \leq k-1$.*
4. *A job J_j is executed during the interval $[t_i, t_{i+1})$, for any $1 \leq i \leq k-1$, if it has been assigned the maximum speed among the speeds of the jobs $J_{j'}$ with $[t_i, t_{i+1}) \subseteq [r_{j'}, d_{j'})$.*

Theorem 1 is also satisfied by the optimal schedule of $1|pmtn, r_j|\sum U_j(E)$ for the jobs in J^*. Specifically, if we know somehow the subset of jobs executed in an optimal schedule for $1|pmtn, r_j|\sum U_j(E)$, then we may construct the actual optimal schedule by using an optimal algorithm for $1|pmtn, r_j, d_j|E$.

4 Optimal Algorithms

For the problem $1|r_j, agreeable|\sum U_j(E)$ where the deadlines of the jobs are agreeable, we propose an optimal algorithm based on dynamic programming. As mentioned before, among the schedules of maximum throughput, our algorithm constructs a schedule of minimum energy consumption. Next, we describe our dynamic program and we elaborate on the complexity of our algorithm.

Let FS be the problem of maximizing the throughput when each job has a fixed processing time, as described in the subsection with the related work. Next, we consider another problem which we denote as CS. In this problem, we are given a set of jobs $J = \{J_1, J_2, \ldots, J_n\}$, where each job J_j has a processing requirement p_j, a release date r_j and a deadline d_j, that have to be executed by a single speed scalable processor. Moreover, we are given a value of throughput k. The objective is to find the minimum energy schedule which completes at least k jobs on time and all jobs run with equal speed. For notational convenience, we denote the problem $1|pmtn, r_j|\sum U_j(E)$ as SS.

The inspiration for our dynamic programming for the special case of the SS where the deadlines are agreeable was the fact that the problem CS can be solved in polynomial time by repeatedly solving instances of the problem FS. In fact, if we are given a candidate speed s for the CS problem, we can find a schedule of maximum throughput w.r.t. to s simply by setting the processing time of each job J_j equal to $\frac{p_j}{s}$ and applying an optimal algorithm for the FS problem. So, in order to get an optimal algorithm of the CS problem, it suffices to establish a lower and upper bound on the speed of the optimal schedule. A naive choice is $s_{min} = 0$ and $s_{max} = \sum_{j=1}^{n} p_j/(d_{mam} - r_{min})$. Note that, r_{min} and r_{max} are the minimum and maximum release date among all jobs, respectively, Similarly, let d_{min} and d_{max} be the minimum and the maximum deadline. Then, it suffices to binary search in $[s_{min}, s_{max}]$ and find the minimum speed s^* in which k jobs are completed on time.

The next property comes from the fact that the algorithm in [14] is optimal for $1|pmtn, r_j, d_j|E$ and it is a key ingredient for decomposing our problem $1|r_j, agreeable|\sum U_j(E)$.

Property 1 There is always an optimal schedule for $1|pmtn, r_j, d_j|E$ in which the jobs are executed according to the EDF (Earliest Deadline First) policy. In the case where the jobs have agreeable deadlines, the jobs are executed without preemptions.

In what follows, we assume that the jobs $J_1, J_2, \ldots, J_n$ are sorted according to the EDF order, i.e. $d_1 \leq d_2 \leq \ldots \leq d_n$.

4.1 Equal Release Dates

Now, we present our dynamic algorithm for the problem $1||\sum w_j U_j(E)$ in which all jobs have the same release date at $t = 0$.

Before giving the formal definition of our algorithm, we give an intuitive explanation on how it works. Since we want a minimum energy schedule among all schedules of maximum throughput, our algorithm does not execute any of the jobs which are not completed on time. Let $J^* \subseteq J$ be the set of jobs which are completed on time by the algorithm. Then, the optimal schedule for $1||\sum w_j U_j(E)$ is a minimum energy schedule for the jobs in J^*.

It has to be noticed that a schedule of a processor whose speed is varied can be represented as a 2-dimensional diagram where the horizontal axis corresponds to the time horizon and the vertical axis corresponds to the speed. Then, by Theorem 1, a minimum energy schedule for the jobs in J^* is a downward staircase as depicted in the following Figure 1.

In the above figure which corresponds to an optimal solution for our problem, we assume that u jobs are executed in total and that job J_k is the last job which is completed on time. The energy consumption of this schedule is denoted as $E(k, u)$. It is possible that all jobs are executed with the same speed but this case can be treated easily. So, we assume for the moment that there is at least a couple of jobs which are executed with different speeds.

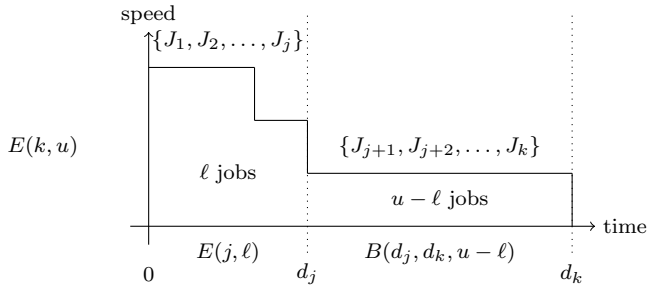

Fig. 1 Illustration of an optimal solution when jobs have common release date

By Theorem 1, the optimal schedule is a downward staircase and every step starts and ends in one of the points $t_1, t_2, \ldots, t_k$. Another important consequence of Theorem 1 is that, for any step between d_j and d_k, only jobs in the set $\{J_{j+1}, J_{j+2}, \ldots, J_k\}$ are allowed to be executed during $[d_j, d_k)$. Moreover, none of these jobs is executed before d_j or after d_k. Therefore, if we knew somehow that $[d_j, d_k)$ is the last step of the optimal schedule and that exactly $u - \ell$ jobs are executed during this step, then we could decompose the problem into the sub-problems $E(j, \ell)$ and $B(d_j, d_k, u - \ell)$, where $B(d_j, d_k, u - \ell)$ is the minimum energy consumption for scheduling exactly $u - \ell$ jobs among $\{J_{j+1}, J_{j+2}, \ldots, J_k\}$ with the same speed during the time interval $[d_j, d_k)$. This latter problem is a CS problem which we know how to solve in polynomial time. Our dynamic programming algorithm is based on this decomposition scheme.

We, now, describe formally our algorithm. For a subset of jobs $S \subseteq \mathcal{J}$, a schedule which involves only the jobs in S will be called a S-schedule.

Definition 1 Let $J(k) = \{J_j \mid j \leq k\}$ be the set of the first k jobs according to the EDF order. For $1 \leq u \leq |J(k)|$, we define $E(k, u)$ as the minimum energy consumption of an S-schedule such that $|S| = u$ and $S \subseteq J(k)$. If such a schedule does not exist, i.e. when $u > |J(k)|$, then $E(k, u) := +\infty$.

Definition 2 We define $B(t', t, \ell)$ as the minimum energy consumption of an S-schedule such that $|S| = \ell$, $S \subseteq \{J_j \mid t' < d_j \leq t\}$ and such that all these jobs are scheduled only within the interval $[t', t]$, and with a constant common speed. If such a schedule does not exist, then $B(t', t, \ell) := +\infty$.

Given an energy budget E, the maximum number of jobs that can be scheduled without exceeding this budget is given by $\max\{u \mid E(n, u) \leq E\}$.

Proposition 1 *$B(d_j, d_k, \ell)$ can be computed in $O(n \log n \log P)$ time, for any j, k, ℓ, with $P = \sum_i p_i$.*

Proof In order to compute $B(d_j, d_k, \ell)$, we consider the set of jobs $\{J_i \mid d_j < d_i \leq d_k\}$. For each job in this set, we modify its release date to d_j. Since we

want the minimum energy consumption and there is only one speed, we search the minimum speed such that there are exactly ℓ jobs scheduled. Moreover, we schedule an integer volume of jobs, then the speed is necessarily $h/(d_k - d_j)$ for some $h = 0, \ldots, P$. This minimum speed can be found by performing a binary search on the value of h in the interval $[0, P]$. Once the value of h is chosen, the processing time of a job J_i is $t_i = p_i \times (d_k - d_j)/h$, and we compute the maximum number m of jobs which can be scheduled using Moore's algorithm [12] in time $O(n \log n)$. If $m < \ell$ (resp. $m > \ell$) the value h must be increased (resp. decreased). □

Proposition 2 *It holds that*

$$E(k, u) = \min_{\substack{0 \leq j \leq k \\ 0 \leq \ell \leq u}} \{E(j, \ell) + B(d_j, d_k, u - \ell)\}.$$

Proof Let S be an optimal schedule associated with $E(k, u)$. We can assume that this schedule satisfies the properties of Theorem 1 and Property 1. Clearly, by Definition 1, $E[0, 0] = 0$ which corresponds to the empty schedule and $E[0, u] = +\infty$, for every $u > 0$, because it is not possible to schedule $u > 0$ number of jobs when there are no jobs available.

If $J_k \notin S$, then $E(k, u) = E(k - 1, u)$. If $J_k \in S$, then there are two cases to consider. The first case is when all the jobs in S are scheduled at the same speed. This case is equivalent to the CS problem, and one has $E(k, u) := B(0, d_k, u)$.

The second case is when the schedule S has at least two different speeds. Let C_j be the completion time of job J_j in the schedule S. Let $t = \min_j\{C_j \mid$ all the jobs scheduled after J_j (at least one job) are executed with the same speed$\} = C_{j^*}$. Necessarily, job J_{j^*} is executed with a different speed. This means that at time C_{j^*} the processor is changing its speed, and using Property 3 of Theorem 1 we can deduce that $C_{j^*} = d_{j^*}$. Now we consider the subschedule S_1 obtained from S by considering only the tasks executed during the interval $[0, d_{j^*})$. Let us assume that there are ℓ^* tasks in this subschedule. Then, necessarily the energy consumption of S_1 is equal to $E(j^*, \ell^*)$, otherwise by replacing S_1 with a better subschedule with energy consumption $E(j^*, \ell^*)$ we could obtain a better schedule than S. Now we consider the subschedule S_2 obtained from S by considering only the tasks executed from time d_{j^*} until the end of the schedule. In a similar way, the energy consumption of S_2 is equal to $B(d_{j^*}, d_k, u - \ell^*)$. Notice that since the jobs involved in $E(j, \ell)$ have a deadline smaller or equal to d_j, whereas the jobs involved in $B(d_j, d_k, u - l)$ have a deadline greater than d_j, those sets of jobs are always distinct, and therefore the schedule associated with $E(j, \ell) + B(d_j, d_k, u - \ell)$ is always feasible. □

Theorem 2 *The problem* $1||\sum U_j(E)$ *can be solved in* $O(n^4 \log n \log P)$ *time.*

Proof We use a dynamic program based on Proposition 2, with $E(0, u) = +\infty$, $\forall u > 0$. The maximum throughput is equal to $\max\{u \mid E(n, u) \leq E\}$. The number of values $B(d_j, d_k, \ell)$ is $O(n^3)$. They can be precomputed with a total

running time $O(n^4 \log n \log P)$, using Proposition 1. The number of values $E(k,u)$ is $O(n^2)$, and the complexity to calculate each $E(k,u)$ value is $O(n^2)$ (we have to look for $O(n^2)$ values for j, ℓ and we assume that the previous $E(.,.)$ values have already been computed). Thus the overall complexity is $O(n^4 \log n \log P)$. □

4.2 Agreeable Instances

Definition 3 We define $E_k(t,u)$ as the minimum energy consumption of an S-schedule, such that $|S| = u$, $S \subseteq J(k,t) = \{J_j \mid j \leq k, r_j < t\}$ and such that all these jobs are executed within the interval $[r_{min}, t]$. If such a schedule does not exist, then $E_k(t,u) = +\infty$.

Definition 4 We define $A(t', t, \ell, j, k)$ as the minimum energy consumption of a S-schedule such that $|S| = \ell$, $S \subseteq \{J_j, \ldots, J_k\}$, and such that all these jobs are scheduled within the interval $[t', t]$, and with a constant common speed.

Proposition 3 *$A(t', t, \ell, j, k)$ can be computed in $O(n \log n \log P)$ time, for any t', t, ℓ, j, k.*

Proof In order to compute $A(t', t, \ell, j, k)$, we change the release date of job J_i to t' if $r_i < t'$, and the deadline of job J_i to t if $d_i > t$. The set $\{J_j, \ldots J_k\}$ still has agreeable deadlines. Then we proceed as in the proof of Proposition 1 using a binary search over the interval $[0, s_{max}]$, with $s_{max} = P/(t - t')$. Note that in this case, we use Lawler's algorithm in [10]. □

Proposition 4 *It holds that*

$$E_k(t,u) = \min_{\substack{0 \leq \ell \leq u \\ r_{min} \leq t' \leq t \\ 0 \leq j < k}} \Big\{ E_j(t', \ell) + A(t', t, u - \ell, j+1, k) \Big\}.$$

Proof Let S be an optimal schedule associated with $E_k(t,u)$. We can assume that this schedule satisfies the properties of Theorem 1 and Property 1. Let E' be the right hand side of the equation.

If $J_k \notin S$, then $E_k(t,u) = E_{k-1}(t,u)$. In that case, $t' = t$, $j = k-1$ and $\ell = u$ in the above expression. If $J_k \in S$, then there are two cases to consider. The first case is when the optimal schedule S has one speed. In that case $t' = r_{min}$, $\ell = 0$, $j = 0$ in the above expression. This case is equivalent to the CS problem. We now consider the second case when the optimal schedule S has at least two speeds. We split the schedule S into two subschedules S_1 and S_2 (see the Figure 2).

There exists t' with $r_{min} < t' < t$, such that all the jobs scheduled after t' are scheduled with a common speed, and this is the subschedule S_2. Let $j+1$ be the job which is scheduled after date t in S. We can suppose that each job in S which release date is before the release date of job $j+1$ are entirely scheduled before t'. If it is not the case, then there is a job $i < j+1$ which

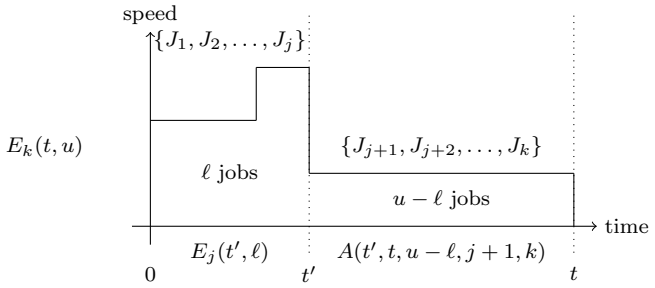

Fig. 2 Illustration of the decomposition in Proposition 4

is scheduled after $j+1$ and we have a contradiction since we only consider EDF schedule. Otherwise there is a job $i < j+1$ which is not entirely executed before t' and we have a preemption of the job i which contradicts Property 1. Thus we have $S_1 \subseteq \{1,\ldots,j\}$ and $S_2 \subseteq \{j+1,\ldots,k\}$.

The restriction S_1 of S to $[0,t')$ is a schedule that meets all constraints related to $E_j(t',\ell)$. Hence its cost is greater than $E_j(t',\ell)$. Similarly, the restriction S_2 of S to $[t',t)$ is a schedule that meets all constraints related to $A(t',t,u-\ell,j+1,k)$. Hence its cost is greater than $A(t',t,u-\ell,j+1,k)$. Thus, $E_k(t,u) \geq E_j(t',\ell) + A(t',t,u-\ell,j+1,k)$. □

Theorem 3 *The problem* $1|r_j, agreeable|\sum U_j(E)$ *can be solved in* $O(n^6 \log n \log P)$ *time.*

Proof We use a dynamic program based on Proposition 4. Notice that the important dates are included in the set $T = \{r_j \mid 1 \leq j \leq n\} \cup \{d_j \mid 1 \leq j \leq n\}$. This comes from Property 1 and Theorem 1, i.e. the changes of speed of the processor occur only at some release date or some deadline. Therefore we can always assume that $t', t \in T$. Notice also that $|T| = O(n)$.

We define $E_0(t,0) = 0\ \forall t \in T$, and $E_0(t,u) = +\infty\ \forall u > 0, t \in T$. The maximum throughput is equal to $\max\{u \mid E_n(d_{max},u) \leq E\}$.

The number of values $A(t',t,\ell,j,k)$ is $O(n^5)$. They can be precomputed with a total processing time $O(n^6 \log n \log P)$, using Proposition 3. The number of values $E_k(t,u)$ is $O(n^3)$. To compute each value, we have to look for the $O(n^3)$ cases (for each value of t', j, ℓ). In each case, we pick up two values which are already computed. Thus the $E_k(t,u)$ values are computed in $O(n^6)$ time. The overall complexity is $O(n^6 \log n \log P)$. □

4.3 Weighted Throughput

Next we consider the weighted version of our problem, i.e. $1|pmtn, r_j|\sum_j w_j U_j(E)$. In this version a job J_j is defined by its release date r_j, its deadline d_j, its amount of work p_j and its weight w_j. We

want to maximize the total weight of the jobs scheduled subject to E. We first show that the problem is $\mathcal{NP}$-hard even in the case where all the jobs are released at the same time and have equal deadlines. Then, we present a pseudo-polynomial algorithm for the case where the deadlines are agreeable.

Theorem 4 *The problem* $1||\sum_j w_j U_j(E)$ *is* $\mathcal{NP}$*-hard.*

Proof In order to establish the $\mathcal{NP}$-hardness of $1||\sum_j w_j U_j(E)$, we present a reduction from the KNAPSACK problem which is known to be $\mathcal{NP}$-hard. In an instance of the KNAPSACK problem we are given a set I of n items. Each item $i \in I$ has a value v_i and a capacity c_i. Moreover, we are given a capacity C, which is the capacity of the knapsack, and a value V. In the decision version of the problem we ask whether there exists a subset $I' \subseteq I$ of the items of total value not less than V, i.e. $\sum_{i \in I'} v_i \geq V$, whose capacity does not exceed the capacity of a knapsack, i.e. $\sum_{i \in I'} c_i \leq C$.

Given an instance of the KNAPSACK problem, we construct an instance of $1||\sum_j w_j U_j(E)$ as follows. For each item i, $1 \leq i \leq n$, we introduce a job J_i with $r_i = 0$, $d_i = 1$, $w_i = v_i$ and $p_i = c_i$. Moreover, we set the budget of energy equal to $E = C^\alpha$.

We claim that the instance of the KNAPSACK problem is feasible iff there is a feasible schedule for $1||\sum_j w_j U_j(E)$ of total weighted throughput not less than V.

Assume that the instance of the KNAPSACK is feasible. Therefore, there exists a subset of items I' such that $\sum_{i \in I'} v_i \geq V$ and $\sum_{i \in I'} c_i \leq C$. Then we can schedule the jobs in I' with constant speed $\sum_{i \in I'} c_i$ during $[0, 1]$. Their total energy consumption of this schedule is no more that C^α since the instance of the Knapsack is feasible. Moreover, their total weight is no less than V.

For the opposite direction of our claim, assume there is a feasible schedule for $1||\sum_j w_j U_j(E)$ of total weighted throughput not less than V. 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 during the whole interval $[0, 1]$ is also feasible. Since the latter schedule is feasible, we have that $\sum_{j \in J'} p_j \leq C$. Moreover, $\sum_{j \in J'} w_j \geq V$. Therefore, the items which correspond to the jobs in J' form a feasible solution for the KNAPSACK. □

In this part, we propose a pseudo-polynomial time algorithm based on a dynamic programming algorithm for the KNAPSACK problem.

Definition 5 We redefine $E_k(t, w)$ to be the minimum energy consumption of a S-schedule, with $S \subseteq J(k, t) = \{J_j \mid j \leq k, r_j < t\}$, such that all the jobs in S are scheduled within the interval $[r_{min}, t]$ and such that the sum of their weight is at least w. If such a schedule does not exist, then $E_k(t, w) = +\infty$.

We redefine $A(t', t, w, j, k)$ to be the minimum energy consumption of a S-schedule such that $S \subseteq \{J_j, \ldots, J_k\}$, $w(S) \geq w$ and such that these jobs are scheduled within the interval $[t', t]$, and with a constant common speed.

Proposition 5 *$A(t', t, w, j, k)$ can be computed in $O(nW \log P)$ time, where W is the sum of weights of the jobs.*

Proof The proof is similar to Proposition 3. In this case, we use Lawler's algorithm in [9]. □

Proposition 6 *It holds that*

$$E_k(t, w) = \min_{\substack{0 \leq \ell \leq w \\ r_{min} \leq t' \leq t \\ 0 \leq j < k}} \Big\{ E_j(t', \ell) + A(t', t, w - \ell, j + 1, k) \Big\}.$$

Proof The proof is similar to Proposition 4. □

Theorem 5 *The problem $1|r_j, agreeable| \sum_j w_j U_j(E)$ can be solved in $O(n^5 W^2 \log P)$ time.*

Proof We use a dynamic program based on Proposition 5, with $E_0(t, 0) = 0$ $\forall t \in T$ and $E_0(t, w) = +\infty$ $\forall w > 0, t \in T$. The maximum weighted throughput is obtained with $\max\{w \mid E_n(d_{max}, w) \leq E\}$. The number of values $A(t', t, \ell, j, k)$ is $O(n^4 W)$. They can be precomputed and finally it takes $O(n^5 W^2 \log P)$ time. The number of values $E_k(t, u)$ is $O(n^2 W)$. To compute each value, we have to look for the $O(n^2 W)$ cases (for each value of t', j, ℓ). In each case, we pick up two values which are already computed. Thus the $E_k(t, u)$ values are computed in $O(n^4 W^2)$ time. Thus the overall complexity is $O(n^5 W^2 \log P)$. □

5 Future Work

While the throughput maximization problem is polynomially-time solvable for agreeable deadlines its complexity remains open for general instances. This is a challenging open question for future research.

References

1. Angel, E., Bampis, E., Chau, V.: Throughput maximization in the speed-scaling setting. In: E.W. Mayr, N. Portier (eds.) STACS, *LIPIcs*, vol. 25, pp. 53–62. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2014)
2. Bansal, N., Chan, H.L., Lam, T.W., Lee, L.K.: Scheduling for speed bounded processors. In: L. Aceto, I. Damgård, L.A. Goldberg, M.M. Halldórsson, A. Ingólfsdóttir, I. Walukiewicz (eds.) ICALP (1), *Lecture Notes in Computer Science*, vol. 5125, pp. 409–420. Springer (2008)
3. Baptiste, P.: 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). DOI 10.1016/S0167-6377(98)00045-5. URL http://dx.doi.org/10.1016/S0167-6377(98)00045-5
4. Chan, H.L., Chan, W.T., Lam, T.W., Lee, L.K., Mak, K.S., Wong, P.W.H.: Energy efficient online deadline scheduling. In: N. Bansal, K. Pruhs, C. Stein (eds.) SODA, pp. 795–804. SIAM (2007)

5. Chan, H.L., Lam, T.W., Li, R.: Tradeoff between energy and throughput for online deadline scheduling. In: K. Jansen, R. Solis-Oba (eds.) WAOA, *Lecture Notes in Computer Science*, vol. 6534, pp. 59–70. Springer (2010)
6. Chan, J.W.T., Lam, T.W., Mak, K.S., Wong, P.W.H.: Online deadline scheduling with bounded energy efficiency. In: J. yi Cai, S.B. Cooper, H. Zhu (eds.) TAMC, *Lecture Notes in Computer Science*, vol. 4484, pp. 416–427. Springer (2007)
7. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of discrete mathematics **5**, 287–326 (1979)
8. Lam, T.W., Lee, L.K., To, I.K.K., Wong, P.W.H.: Energy efficient deadline scheduling in two processor systems. In: T. Tokuyama (ed.) ISAAC, *Lecture Notes in Computer Science*, vol. 4835, pp. 476–487. Springer (2007)
9. Lawler, E.L.: A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. Annals of Operations Research **26**(1), 125–133 (1990)
10. Lawler, E.L.: Knapsack-like scheduling problems, the moore-hodgson algorithm and the tower of sets property. Mathematical and Computer Modelling **20**(2), 91–106 (1994)
11. Li, M.: Approximation algorithms for variable voltage processors: Min energy, max throughput and online heuristics. Theor. Comput. Sci. **412**(32), 4074–4080 (2011)
12. Moore, J.M.: An n job, one machine sequencing algorithm for minimizing the number of late jobs. Management Science **15**(1), 102–109 (1968)
13. Pruhs, K., Uthaisombut, P., Woeginger, G.J.: Getting the best response for your erg. ACM Transactions on Algorithms **4**(3) (2008). DOI 10.1145/1367064.1367078. URL http://doi.acm.org/10.1145/1367064.1367078
14. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced CPU energy. In: FOCS, pp. 374–382. IEEE Computer Society (1995)