Non-preemptive Throughput Maximization for Speed-Scaling with Power-Down

Eric Angel[1], Evripidis Bampis[2], Vincent Chau[3] and Nguyen Kim Thang[1]

[1] IBISC ; Université d'Évry Val d'Essonne, Évry, France
[2] Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, Paris, France
[3] Department of Computer Science, City University of Hong Kong, Hong Kong, China

Abstract. We consider the problem of scheduling a set of n jobs on a single processor. Each job is characterized by its release date r_j, its deadline d_j and its processing volume p_j. The processor can vary its speed and can switch into a sleep state in order to reduce its energy consumption. No energy is consumed in this state, but a fixed amount of energy, equal to L, is required for a transition from the sleep state to the active state. Here, we study the *throughput maximization* version of the problem where we are given a budget of energy E and our goal is to determine a feasible schedule maximizing the number of jobs that are executed between their respective release dates and deadlines without preemption. We first consider the case in which jobs have agreeable deadlines, i.e. for every pair of jobs i and j, one has $r_i \leq r_j$ if and only if $d_i \leq d_j$. Then we consider the case where the jobs have arbitrary release dates and deadlines, but the same processing volume. We propose polynomial-time algorithms for both cases.

1 Introduction

Power management aims to reduce the energy consumption in computer systems while maintaining a good level of performance. One of the mechanisms used to save energy is speed-scaling where the processor is capable to vary its speed dynamically. The faster the processor runs, the more it consumes energy. Another mechanism for energy savings is the power-down mechanism, in which we have to decide whether to put the system into the sleep state when it is idle, or maintain it in the active state. No energy is consumed during the sleep state, but a fixed amount of energy is required to wake up the system.

More formally, we are given a set of n jobs. Each job is characterized by its release date r_j, its deadline d_j and its processing volume p_j. We are also given one processor which can vary its speed and can switch into the sleep state in order to reduce the energy consumption. No energy is consumed in this state, but a fixed amount of energy is required for transitioning the system from the sleep state to the active one which is equal to L. The processor can execute at most one job at each time during the active state. We measure the processor's speed in units of executed work per unit of time. If $s(t)$ denotes 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 $\int_t^{t'} s(u)du$. Specifically, at any time t, the power consumption of the processor is $P(t) = s(t)^\alpha + \gamma$, where $\alpha > 1$ is a constant and γ is the energy leakage. We are given an energy budget E and the goal is to find a feasible schedule maximizing the number of jobs that are executed between their release dates and their deadlines and respecting the energy budget.

Related works

A series of papers have studied *energy minimization* for the speed-scaling model, the power-down model and the combined speed-scaling and power-down model.

Speed-scaling. The energy minimization problem of scheduling n jobs with release dates and deadlines on a single processor that can vary its speed dynamically and where the preemption of the jobs is allowed has been first studied in the seminal paper by Yao et al. [22]. The time complexity for general instances has been improved in [20] and [15], while for jobs with agreeable deadlines in [21] and [16]. For the non-preemptive energy minimization problem, Antoniadis and Huang [7] showed its NP-hardness in the strong sense and they proposed a polynomial time approximation algorithm. This result has been improved recently in [9] and [14].
Power-down. Baptiste [10] showed that the problem is polynomially solvable for the power-down model when the processor's speed is fixed by introducing $O(n^7)$ time algorithm. Later on, Baptiste et al. [11] improved the time complexity of this algorithm to $O(n^5)$. When the jobs have agreeable deadline (a formal definition is given below), an $O(n^2)$ exact algorithm has been proposed by Angel et al. [3]. Finally, Chrobak et al. [13] proposed a 2-approximation algorithm of complexity $O(n^2 \log n)$.
Speed-scaling and Power-down. Irani et al. [17] introduced a model incorporating both mechanisms, i.e. speed-scaling and power-down, and proposed a 2-approximation algorithm for the speed-scaling with power-down model. Recently, Albers et al. [1] proved that the speed-scaling problem with power down is NP-hard and provided a 4/3-approximation algorithm for the preemptive case, while Bampis et al. [8] showed that when jobs have agreeable deadlines, the problem can be solved in $O(n^3)$ time.

The most relevant works in the *throughput maximization* version of this problem but only for the speed scaling settings are [19] [5] and [4]. Li [19] has considered throughput maximization when there is an upper bound in the processor's speed and he proposed a greedy algorithm which leads to a 3-approximation for the throughput and a constant approximation ratio for the energy consumption. Angel et al. studied the throughput maximization problem in [5]. They proposed a polynomial time algorithm that solves optimally the single-processor problem for agreeable instances. More recently in [4], the authors proved that there is a pseudo-polynomial time algorithm for solving optimally the preemptive throughput maximization problem with arbitrary release dates and deadlines as well as

arbitrary processing volumes. For the weighted version, the problem is NP-hard even for instances in which all the jobs have common release dates and deadlines.

Our Contributions

We consider the non-preemptive throughput maximization problem for the speed-scaling and power-down model. Notice that this problem is at least as hard as the energy minimization version since we can make a binary search in the energy budget in order to find the minimum energy consumption for all jobs with respect to the accuracy of the search. Since the non-preemptive general case is strongly NP-hard [7] for the speed-scaling settings, then it is also for the combined model. Indeed, the decision version is to ask whether there exists a non-preemptive schedule that have a cost less than some value, and if so, we could construct a solution of the same cost for the 3-partition. Here, we prove that the problem can be solved in polynomial time for two special cases of instances. First, we consider instances where jobs have agreeable deadlines, i.e. such that for every pair of jobs $i < j$, one has $r_i \leq r_j$ if and only if $d_i \leq d_j$. Intuitively, in an instance with agreeable deadlines a later released job also has a later deadline. We propose a $O(n^{11}$ time algorithm to solve this case. Then we consider the case where jobs have arbitrary release dates and deadlines, but have the same processing volume, i.e. $p_j = p \ \forall j$. We first prove that the complexity time remains the same as for the speed scaling case, and improve the time complexity of the energy minimization variant to $O(n^{19})$. These families of instances have received a lot of attention in the literature (see for instance [2], [16]).

2 Preliminaries

In this paper, we consider schedules without preemption. Without loss of generality, we assume that all parameters of the problem such as release dates, deadlines and processing volumes of jobs are integers. 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, i.e. $\Omega := \{r_j | j = 1, \ldots, n\} \cup \{d_j | j = 1, \ldots, n\}$.

We refer to the energy cost of the processor induced by the speed as dynamic energy and to the energy induced by the leakage and the wake-up cost as static energy.

We call an EDF (Earliest Deadlines First) schedule, a schedule in which at any time, the job with the smallest deadline among the available jobs is scheduled first.

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 $P(s) := s^\alpha + \gamma$, each job runs at a constant speed during its whole execution in an optimal schedule. This follows from Jensen's Inequality.

We first define the critical speed that minimizes the energy consumption of a single job. This shows that we do not scheduled every job at the lowest speed as possible.

Definition 1. *[17] Let* $s^\star := \arg\min_s \frac{P(s)}{s}$ *be the critical speed.*

Definition 2. *We define a block of jobs with respect to a schedule as the maximal interval when the processor is in the working state (ON-state). The processor is in the sleep state (OFF-state) before and after this block.*

Definition 3. *(Figure 1) [8] We define the prefix (resp. suffix) of a block of jobs with respect to a schedule as the maximal subset of continuous scheduled jobs at the beginning (resp. at the end) of the block and such that the processor runs at speed* $s^\star$*.*

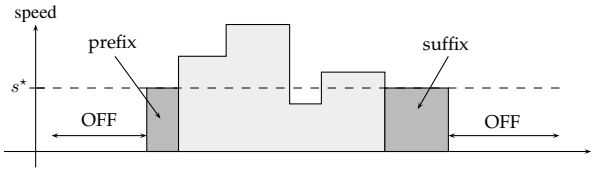

Fig. 1. Illustration of Definition 3

Remark 1. The prefix and suffix of a block of jobs can be empty.

Proposition 1 *[8] The processor speed is at least* $s^\star$ *at the beginning and at the end of a continuous block of jobs.*

Proof. Let S be an optimal schedule and suppose that there is a block B of jobs such that the processor speed is strictly lower than $s^\star$ at the beginning of the block (prefix). Let j be the first job of B. Job j is therefore scheduled with a speed $s < s^\star$ and let $\ell(j)$ be its execution length. The energy consumption of job j is $\ell(j) \cdot P(s)$. Then we can increase the processor speed from s to $s^\star$ for this job, the length of the job b will decrease to $\frac{\ell(j)\cdot s}{s^\star}$, by shifting to the right its starting time without changing its finishing time, the overall schedule will remain feasible, and its energy consumption will decrease. Indeed, the new energy consumption of job j is $P(s^\star) \cdot \frac{\ell(j)\cdot s}{s^\star}$, and one has $\ell(j) \cdot P(s) > P(s^\star)\frac{\ell(j)\cdot s}{s^\star}$ which follows immediately from the definition of the critical speed. □

3 Agreeable Deadlines Jobs

We study in this part instances in which jobs have agreeable deadlines, i.e. for every pair of jobs i and j, one has $r_i \leq r_j$ if and only if $d_i \leq d_j$.

Proposition 2 *There exists an optimal solution in which jobs are scheduled according to the* EDF *order and without preemption.*

Proof. We prove this proposition in two part. The first part is to prove that jobs are scheduled in EDF order. Then we prove that the schedule can be transform into a non-preemptive schedule. Let $\mathcal{O}$ be an optimal schedule in which there exists two consecutive piece of jobs $j' < j$ such 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. Once we obtain a schedule in which jobs are in the EDF order, we may have a same job into several pieces. If the processor is at sleep state between two pieces, then we can merge the second piece with the first one by shifting the piece to the left. The energy's consumption does not increase. Similarly, if the processor is on active state between the two pieces, then we can shift the second piece to the left without increasing the energy's consumption. □

Definition 4. *Define $Y(i, j, a, b, u)$ as the minimum energy consumption of a T-schedule $\mathcal{T}$ such that:*

- *$T \subseteq \{i, \ldots, j\}$ and $|T| = u$ where T is the set of jobs scheduled in $\mathcal{T}$,*
- *the jobs of T are entirely scheduled in $[a, b)$,*
- *the processor is in working state (ON-state) during $[a, b)$.*

Remark 2. By convention, $\{i, i-1\}$ is an empty set.

The energy consumption of a schedule corresponding to $Y(i, j, a, b, u)$ is exactly the minimum combined dynamic and static energy. Note that $Y(i, j, a, b, u)$ could be computed with the algorithm proposed in [5]. Note that by applying this algorithm, we obtain a non-preemptive schedule.

Definition 5. *Define $X(i, j, a, b, u)$ as the minimum energy consumption of a T-schedule $\mathcal{T}$ such that:*

- *$T \subseteq \{i, \ldots, j\}$ and $|T| = u$ where T is the set of jobs scheduled in $\mathcal{T}$,*
- *during interval $[a, b)$, there is* exactly *one sub-interval where the processor is in sleep state. Otherwise, the processor executes jobs in T with speed (exactly) s^*.*

By definition, the energy consumption of the schedule is $((a' - a) + (b - b'))P(s^\star) + L$ where $[a', b') \subseteq [a, b)$ is the interval during which the processor is in sleep state. Note that the processor needs to be waken up once, which costs L. Intuitively, $X(i, j, a, b, u)$ represents the energy consumption during an interval which consists of a prefix and a suffix and a sleep-state interval. Since the speed is fixed in the definition of $X(.)$, we show how to compute the minimum energy consumption with classical scheduling algorithm.

Definition 6. *We define $C(i, j, a, b, u)$ (resp. $S(i, j, a, b, u)$) as the minimum length of a schedule starting at time a (resp. ending at time b), such that there are exactly u jobs among $\{i, \ldots, j\}$ which are scheduled, and such that all these jobs are scheduled in $[a, b)$ with the speed $s^\star$. If no such schedule exists, then the length is infinity.*

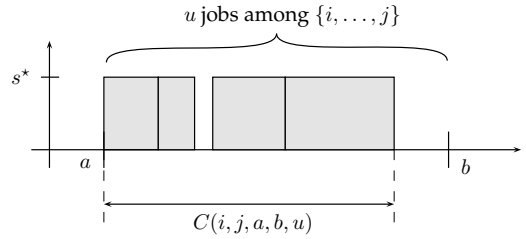

Fig. 2. Illustration of the definition of $C(i, j, a, b, u)$ in Definition 6

Proposition 3 *The values $C(i, j, a, b, u)$ and $S(i, j, a, b, u)$ can be computed in polynomial time.*

Proof. This can be done by setting:

- the processing time of each job k, $i \leq k \leq j$, to $p_k/s^\star$,
- the release date r_k of a job k to $\max\{r_k, a\}$ for $i \leq k \leq j$,
- the deadline d_k of a job k to $\min\{d_k, b\}$ for $i \leq k \leq j$.

Note that with these modifications, we still have an agreeable instance. Finally, we solve this problem with the algorithm proposed in [18]. Indeed, in the classical scheduling problem, the throughput maximization can be reduced to the minimum makespan problem [12], so $C(i, j, a, b, u)$ is computed correctly and in time $O(n \log n)$ if all parameters i, j, a, b, u are set.

Similarly, $S(i, j, a, b, u)$ is the minimum length of a schedule finishing at time b, such that there are exactly u jobs among $\{i, \ldots, j\}$ which are scheduled, and such that all these jobs are scheduled in $[a, b)$ with the speed $s^\star$. This value can be computed in a similar way as $C(.)$ values and by reversing the schedule. More formally, we set b as the starting time of the schedule. Then, for each job k with $i \leq k \leq j$:

- if $r_k \leq b$, we set $d_k^* := b + (b - r_k) = 2b - r_k$;
- if $d_k \leq b$, we set $r_k^* := b + (b - d_k) = 2b - d_k$.

Finally, we solve the minimum makespan problem with the modified jobs with the same algorithm as previously. Thus, it computes the values $S(i, j, a, b, \ell)$ correctly and in time $O(n \log n)$ if all parameters i, j, a, b, u are set. □

Proposition 4 *It holds that*

$$X(i,j,a,b,u) = L + P(s^\star) \cdot \min_{\substack{i-1 \le k \le j \\ 0 \le \ell \le u \\ C := C(i,k,a,b,\ell) \\ S := S(k+1,j,a,b,u-\ell) \\ C+S \le b-a}} \left\{ \begin{array}{c} C(i,k,a,b,\ell) \\ +S(k+1,j,a,b,u-\ell) \end{array} \right\}$$

Proof. The energy consumption depends on the length of the period that the processor is in working state with a speed $s^\star$, and the above computation returns a feasible schedule minimizing this length. □

Definition 7. *For $0 \le u \le n$, define $E(i,j,a,b,u)$ the minimum energy consumption of a T-schedule $\mathcal{T}$ such that:*

- *$T \subset \{i,\ldots,j\}$ and $|T| = u$ where T is the set of jobs scheduled in $\mathcal{T}$,*
- *the jobs of T are entirely scheduled in $[a,b)$.*

Proposition 5 *One has*

$$E(i,j,a,b,u) = \min \begin{cases} Y(i,j,a,b,u) \\ \min\limits_{\substack{a_1,b_1 \\ 0 \le u_1,u_2 \le u \\ i \le i',j' \le j \\ i'-1 \le j'}} \left\{ \begin{array}{c} Y(i,i'-1,a,a_1,u_1) \\ +X(i',j',a_1,b_1,u_2) \\ +E(j'+1,j,b_1,b,u-u_1-u_2) \end{array} \right\} \end{cases}$$

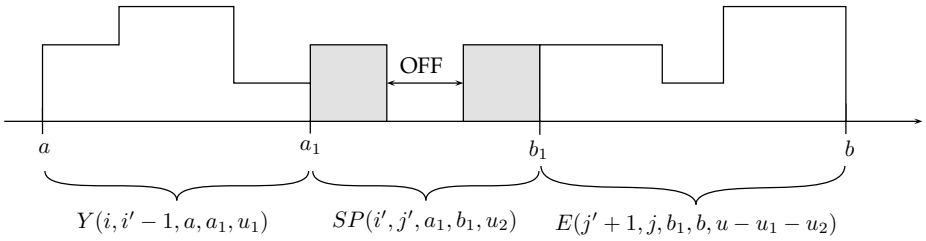

Fig. 3. Illustration of Proposition 5

Proof. (See Figure 3) Let $E' := \min\{E'_1, E'_2\}$ be the right hand side of the equation. The first case is when the processor is never idle during the interval $[a,b)$ and the second is when there is at least one idle period.

We first prove that $E(i,j,s,t,u) \le E'$.
If the processor is never idle, then we can build the schedule associated with $Y(i,j,s,t,u)$ from s to t. Since this schedule respects all the constraints associated with $E(i,j,s,t,u)$ it shows that $E(i,j,s,t,u) \le Y(i,j,s,t,u) = E'_1$.

Suppose now that there is at least one idle period in the schedule. Let a schedule S_1 that realizes $Y(i, i'-1, a, a_1, u_1)$, a schedule S_2 that realizes $X(i', j', a_1, b_1, u_2)$ and a schedule S_3 that realizes $E(j'+1, j, b_1, b, u-u_1-u_2)$. We can build a schedule with S_1 from a to a_1, with S_2 from a_1 to b_1 and with S_3 from b_1 to b. Moreover, the sets of jobs $\{i, \ldots, i'-1\}, \{i', \ldots, j'\}, \{j'+1, \ldots, j\}$ do not intersect. So this is a feasible schedule, and its cost is E'_2. Hence $E(i, j, a, b, u) \leq E'_2$.

We now prove that $E' \leq E(i, j, a, b, u)$.
The first case is that the processor is never idle in a solution. Obviously, we have $E' = Y(i, j, a, b, u)$.

Suppose now that there is at least one idle period in the schedule corresponding to $E(i, j, a, b, u)$. We denote by $\mathcal{X}$ the schedule that realizes $E(i, j, a, b, u)$ in which the first starting time a_1 of the first suffix block is maximal and the first completion time b_1 of the first prefix block is minimal. We split $\mathcal{X}$ into three sub-schedules $\mathcal{S}_1 \subseteq \{i, i'-1\}$, $\mathcal{S}_2 \subseteq \{i', j'\}$ and $\mathcal{S}_3 \subseteq \{j'+1, j\}$ according to the set of jobs executed into those three sub-schedules. Let a' (resp. b') be the completion time (resp. starting time) of the first suffix (resp. prefix) block. We have $a_1 \leq a' < b' \leq b_1$. We claim that we have the following properties:

- the processor execute always something in $[a_1, a')$
- the processor execute always something in $[b', b_1)$

Suppose that the processor does not execute a job at time a'' with $a_1 \leq a'' \leq a'$, then we can take $a_1 := a''$ and we have a contradiction with the fact that a_1 was maximal and we have another job j'. A similar argument can be applied for the value of b_1.

Then the restriction $\mathcal{S}_1$ of $\mathcal{X}$ in $[a, a_1)$ is a schedule that meets all constraints related to $Y(i, i'-1, a, a_1, |\mathcal{S}_1|)$ and its cost is greater than $Y(i, i'-1, a, a_1, |\mathcal{S}_1|)$. Similarly, the restriction $\mathcal{S}_2$ of $\mathcal{X}$ in $[a_1, b_1)$ is a schedule that meets all constraints related to $X(i', j', a_1, b_1, |\mathcal{S}_2|)$. Finally, the restriction $\mathcal{S}_3$ of $\mathcal{X}$ in $[b_1, b)$ is a schedule that meets all constraints related to $E(j'+1, j, b_1, b, |\mathcal{S}_3|)$. Then the cost of the schedule $\mathcal{X}$ is the sum of the cost of the schedule $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$.

Hence $E(i, j, a, b, u) \geq E'_2$. □

Theorem 1. *The dynamic program in Proposition 5 has a running time of* $O(n^{11})$.

Proof. We can precompute the values of $Y(i, j, a, b, u)$. It is sufficient to set the value of i and compute with the algorithm in [5]. Since it is also a dynamic programming, it will compute the other values of $Y(i, j, a, b, u)$. Then the table $Y(i, j, a, b, u)$ can be computed in time $O(n \times n^6 \log V \log n)$.

We can also precompute the values of $X(i, j, a, b, u)$. By setting the values i, j, a, b, u, we have to minimize over the values k and ℓ. Once all the values are set, the algorithm needs $O(n \log n)$ to compute (twice) the minimum makespan. Finally the table $X(i, j, a, b, u)$ can be computed in time $O(n^8 \log n)$.

Finally, the table $E(i, j, a, b, u)$ has size $O(n^5)$. We have to minimize over the values $a_1, b_1, u_1, u_2, i', j'$ where we pick already computed values. Then

the overall time complexity is $O(n^{11})$. Note that the objective function is $\max\{u \mid E(1,n,0,d_n,u) \leq E\}$. □

4 Equal Processing Volume

In this section, we assume that $p_j = p$ for every job j, and we extend the result in [6] to the model with speed-scaling and power-down. Although the results in [6] solve the multiprocessor case, we focus here on the single processor case but our result can be easily extended to the multiprocessor case when the number of processors is a given constant. We first define the set of relevant dates for our problem.

Definition 8. *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,* $\Theta := \bigcup\{\Theta_{a,b} | a,b \in \Omega\} \cup \{a \pm \ell \cdot \frac{p}{s^\star} \mid a \in \Omega \text{ and } \ell = 1,\ldots,n\}$.

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

Lemma 1. *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. $\mathcal{O}$ 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 $1 \leq \ell \leq k$.

The first part of this proof comes from [6] where the the starting and completion times of each job in any sub-blocks $B_2,\ldots,B_{k-1}$ belong to $\Theta_{a,b}$ for some $a \in \Omega$ and $b \in \Omega$.

Finally, thanks to Proposition 1, the speed of sub-block B_1 and B_k are at least $s^\star$. If the speed of the processor during the sub-block B_1 is equal to $s^\star$, then the starting time of the sub-block B_1 is at a distance $h \cdot \frac{p}{s^\star}$ from the completion time of B_1 (which is a date in Ω). Thus the starting time of B_1 is in Θ. Note that there are h jobs in B_1 and each job has its starting time and completion time in $\{a \pm \ell \cdot \frac{p}{s^\star} \mid a \in \Omega \text{ and } \ell = 1,\ldots,n\}$. If the speed of the processor during the sub-block B_1 is strictly larger to $s^\star$, then the starting time of the sub-block B_1 is necessary a release date, otherwise, we can decrease the speed and the energy consumption as well.

Similarly, the completion time of the sub-block B_k is either a deadline, either at a distance $h \cdot \frac{p}{s^\star}$ from the starting time of the sub-block B_k for some value $h = 1,\ldots,n$. □

Using Lemma 1 we can assume that each job is processed at some speed which belongs to the following set.

Definition 9. *Let* $\Lambda := \{\frac{\ell \cdot p}{b-a} \mid \ell = 1,\ldots,n \text{ and } a,b \in \Omega \text{ and } a < b\} \cup \{s^\star\}$ *be the set of different speeds.*

Definition 10. *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.

Definition 11. *For* $0 \leq u \leq n$, *define* $E_k(a,t,b,u)$ *as the minimum energy consumption of a non-preemptive schedule* $\mathcal{S}$ *such that*

- $S \subseteq J(k,a,b)$ *and* $|S| = u$ *where* S *is the set of jobs scheduled in* $\mathcal{S}$,
- *it is idle during interval* $[a,t]$.

Note that $E_k(a,t,b,u) := \infty$ if no such schedule $\mathcal{S}$ exists.

Proposition 6 *(Figure 4) One has*

$$E_0(a,t,b,0) = \gamma(t-a) + \min\{L, \gamma(b-t)\}$$
$$E_0(a,t,b,u) = +\infty \ \forall u \neq 0$$
$$E_k(a,t,b,u) = \min \begin{cases} E_{k-1}(a,t,b,u) \\ \min\limits_{\substack{a',t' \in \Theta \\ s \in \Lambda \\ t'=a'+p/s \\ r_k \leq a' < t' \leq d_k \\ 0 \leq u' \leq u-1}} \left\{ \begin{array}{l} E_{k-1}(a,t,a',u') + p \cdot s^{\alpha-1} \\ +E_{k-1}(a',t',b,u-u'-1) \end{array} \right\} \end{cases}$$

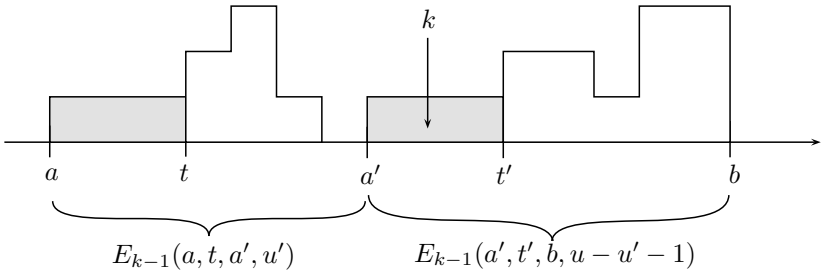

Fig. 4. Illustration of Proposition 6

The main difference with [6] lies in the initialization of the table where we consider the energy consumption of idle periods since there is no jobs scheduled at these moments. The proof of the dynamic program is the same as for the speed-scaling model.

Theorem 2. *The dynamic program in Proposition 6 has a running time of* $O(n^{21})$.

Proof. The objective function is $\max\{u \mid E_n(r_{\min}, r_{\min}, d_n, u) \leq E\}$. The values of $E_k(.)$ are stored in a multi-dimensional array of size $O(|\Theta|^2|\Lambda|n^2)$. Each value

need $O(|\Lambda||\Theta|\ W)$ time to be computed thanks to Proposition 6. Thus we have a total running time of $O(|\Theta|^3|\Lambda|^2n^3)$. This leads to an overall time complexity $O(n^{21})$. □

Corollary 1. *The minimization energy version of the problem can be solved in time $O(n^{19})$.*

Proof. In the minimization energy variant, we have to schedule every job. Then it is not necessary to guess how many jobs are scheduled in each part. We have to redefine the definition of $E_k(a,t,b,u)$ into $E_k(a,t,b)$ where we have to schedule every jobs in $\{j \leq k \mid a \leq r_j < b\}$. Moreover, we don't need to guess the value of u' in the dynamic program. By adapting the dynamic program in Proposition 6, we decrease by $O(n)$ the size of the table, and by $O(n)$ the time complexity to compute one value of the table. Therefore, the energy minimization variant has a complexity of $O(n^{19})$. □

5 Concluding remarks

Maximizing throughput is one of the most important problems in scheduling theory. In the last years, an increasing interest has been devoted to this problem in the energy-aware setting. While the problem is closely related to the energy minimization problem, throughput maximization seems harder to tackle. For instance, while a polynomial time algorithm is known for the preemptive energy minimization problem for the single-processor case, the complexity of the throughput maximization problem remains still open. Only recently, a pseudo-polynomial time algorithm appeared in the literature showing that the problem is not strongly NP-hard [4], but the question of whether the problem is solvable in polynomial time or not remains open. For the speed-scaling with power-down model, the question is even more challenging. While a polynomial time algorithm existed for instances with agreeable deadlines, no results were known for the throughput maximization problem. Our paper closes this gap by showing that the problem is polynomial for these types of instances. For instances with arbitrary release dates and deadlines, the question remains wide open. As a first step in this direction, we showed in this paper that the problem is polynomially time solvable if the processing volumes of the jobs are identical. However, no results for the throughput maximization problem are known for arbitrary processing volumes and arbitrary release dates and deadlines.

References

1. Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. In *SODA*, pages 1266–1285. SIAM, 2012.
2. Susanne Albers, Fabian Müller, and Swen Schmelzer. Speed scaling on parallel processors. In *SPAA*, pages 289–298. ACM, 2007.
3. Eric Angel, Evripidis Bampis, and Vincent Chau. Low complexity scheduling algorithm minimizing the energy for tasks with agreeable deadlines. In *LATIN*, volume 7256 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2012.

4. Eric Angel, Evripidis Bampis, and Vincent Chau. Throughput maximization in the speed-scaling setting. In *STACS*, volume 25 of *LIPIcs*, pages 53–62. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.
5. Eric Angel, Evripidis Bampis, Vincent Chau, and Dimitrios Letsios. Throughput maximization for speed-scaling with agreeable deadlines. In *TAMC*, volume 7876 of *Lecture Notes in Computer Science*, pages 10–19. Springer, 2013.
6. Eric Angel, Evripidis Bampis, Vincent Chau, and Nguyen Kim Thang. Throughput maximization in multiprocessor speed-scaling. *CoRR*, abs/1402.3782, 2014.
7. Antonios Antoniadis and Chien-Chung Huang. Non-preemptive speed scaling. *J. Scheduling*, 16(4):385–394, 2013.
8. Evripidis Bampis, Christoph Dürr, Fadi Kacem, and Ioannis Milis. Speed scaling with power down scheduling for agreeable deadlines. *Sustainable Computing: Informatics and Systems*, 2(4):184 – 189, 2012.
9. Evripidis Bampis, Alexander Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Maxim Sviridenko. Energy efficient scheduling and routing via randomized rounding. In *FSTTCS*, volume 24 of *LIPIcs*, pages 449–460. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
10. Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *SODA*, pages 364–367. ACM Press, 2006.
11. Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial time algorithms for minimum energy scheduling. In *ESA*, volume 4698 of *Lecture Notes in Computer Science*, pages 136–150. Springer, 2007.
12. Peter Brucker. *Scheduling Algorithms*. Springer Publishing Company, Incorporated, 5th edition, 2010.
13. Marek Chrobak, Uriel Feige, Mohammad Taghi Hajiaghayi, Sanjeev Khanna, Fei Li, and Seffi Naor. A greedy approximation algorithm for minimum-gap scheduling. In *CIAC*, volume 7878 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2013.
14. Vincent Cohen-Addad, Zhentao Li, Claire Mathieu, and Ioannis Milis. Energy-efficient algorithms for non-preemptive speed-scaling. *CoRR*, abs/1402.4111, 2014.
15. Bruno Gaujal and Nicolas Navet. Dynamic voltage scaling under EDF revisited. *Real-Time Systems*, 37(1):77–97, 2007.
16. Bruno Gaujal, Nicolas Navet, and Cormac Walsh. Shortest-path algorithms for real-time scheduling of FIFO tasks with minimal energy use. *ACM Trans. Embedded Comput. Syst.*, 4(4):907–933, 2005.
17. Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3(4), 2007.
18. Hiroshi Kise, Toshihide Ibaraki, and Hisashi Mine. A solvable case of the one-machine scheduling problem with ready and due times. *Operations Research*, 26(1):121–126, 1978.
19. Minming Li. Approximation algorithms for variable voltage processors: Min energy, max throughput and online heuristics. *Theor. Comput. Sci.*, 412(32):4074–4080, 2011.
20. Minming Li and Frances F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. In *MFCS*, volume 3618 of *Lecture Notes in Computer Science*, pages 652–663. Springer, 2005.
21. Weiwei Wu, Minming Li, and Enhong Chen. Min-energy scheduling for aligned jobs in accelerate model. *Theor. Comput. Sci.*, 412(12-14):1122–1139, 2011.
22. F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382. IEEE Computer Society, 1995.