ARTICLE IN PRESS

Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines☆

Eric Angel [a], Evripidis Bampis [b], Vincent Chau [a,b,*]

[a] Laboratoire IBISC, Université d'Evry, Evry, France

[b] Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France

ARTICLE INFO

Article history:
Received 17 September 2012
Received in revised form 6 April 2014
Accepted 18 May 2014
Available online xxxx

Keywords:
Scheduling
Power management
Unit tasks
Identical processors

ABSTRACT

Power management aims in reducing the energy consumed by computer systems while maintaining a good level of performance. One of the mechanisms used to save energy is the shut-down mechanism which puts the system into a sleep state when it is idle. No energy is consumed in this state, but a fixed amount of energy is required for a transition from the sleep state to the active state which is equal to L times the energy required for the execution of a unit-time task. In this paper, we focus on the off-line version of this problem where a set of unit-time tasks with release dates and deadlines have to be scheduled in order to minimize the overall consumed energy during the idle periods of the schedule. Here we focus on the case where the tasks have agreeable deadlines. For the single processor case, an $O(n^3)$ algorithm has been proposed in Gururaj et al. (2010) for unit-time tasks and arbitrary L. We improve this result by introducing a new $O(n^2)$ polynomial-time algorithm for tasks with arbitrary processing times and arbitrary L. For the multiprocessor case we also improve the complexity from $O(n^3m^2)$ Gururaj et al. (2010) to $O(n^2m)$ in the case of unit-time tasks and unit L.

1. Introduction

We focus on the following scheduling setting. We are given a set of n tasks: task i is characterized by its processing time p_i, its release date r_i and its deadline d_i. To save energy, we adopt the *power-down mechanism*, which has to decide whether to put the system into the *sleep* state when it is idle, or to maintain it in the active state. No energy is consumed in the sleep state, but a fixed amount of energy is required for a transition from the sleep to the active state which is equal to L times the energy required for the execution of a unit time task. The consumed energy between two consecutive tasks of a given schedule depends on the relation of the length of the idle period between them and the value of L. Let $gap_\sigma(i, j)$ be the length of the idle period between two consecutive tasks i and j in a schedule σ. Then, the consumption of energy for this idle period is equal to $gap_\sigma(i, j)$ if $gap_\sigma(i, j) < L$, otherwise it is equal to L. This means that whenever the idle period between two consecutive tasks has a length less than L then the machine remains in the active state, otherwise it changes to the sleep state. Our aim is to find a feasible schedule σ that executes every task i during its time interval $[r_i, d_i)$ and minimizes the overall consumed energy during the idle periods of the schedule. Denote this energy by $E(\sigma)$. In the following, we consider that the tasks have *agreeable deadlines*, i.e. for every pair of tasks i and j, one has $r_i \leq r_j$ if and only if $d_i \leq d_j$.

☆ This work has been supported by the ANR project TODO (09-EMER-010) and the GDR RO du CNRS.

* Corresponding author.
E-mail address: Vincent.Chau@ibisc.univ-evry.fr (V. Chau).

http://dx.doi.org/10.1016/j.dam.2014.05.023
0166-218X/

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023

Table 1
Summary of results.

Number of proc.	L	p_j	Assumption	time
1	Any	1		$O(n^7)$ [2] $O(n^4)$ [3]
1	Any	Any	Preemption	$O(n^5)$ [3]
m	Any	1		$O(n^7m^5)$ [6]
1	1	Any	Agreeable	$O(n \log n)$ [7]
1	Any	1	Agreeable	$O(n^3)$ [7] $O(n^2)$ [*this paper*]
1	Any	Any	Agreeable	$O(n^2)$ [*this paper*]
m	1	1	Agreeable	$O(n^3m^2)$ [7] $O(n^2m)$ [*this paper*]

Previous results and our contribution. There is an increasing interest in power management policies in the literature, both concerning shut-down mechanisms as the one considered in this paper, or speed scaling mechanisms where the speed (or frequency) of the processor(s) can be changed during the execution. For example, we refer the reader to [1,4,8] for a detailed state-of-the-art on algorithmic problems in power management. However, the most related results are the ones of [5,2] and [3]. Chrétienne [5] proved that it is possible to decide in polynomial time whether there is a schedule with no idle time. Baptiste [2] proposed a $O(n^7)$ polynomial-time algorithm for unit-time tasks and general L. More recently Baptiste, Chrobak and Dürr [3] proposed an $O(n^5)$ polynomial-time algorithm for the general case with tasks of arbitrary lengths where preemptions are allowed. They also proposed an $O(n^4)$ algorithm for unit-time tasks.

Given the high time complexity of the algorithms in the general case, Gururaj et al. [7] improved the time-complexity by restricting their attention to an important family of instances, where the tasks have agreeable deadlines. The agreeable deadline property just means that later released jobs also have later deadlines. This holds for instance, when the deadline of each job is exactly F units after its release date. Such a situation may arise when one wants to maintain a guarantee of service for the flow time of the jobs. They proposed an $O(n \log n)$ greedy algorithm for agreeable deadlines for the single processor case, arbitrary lengths tasks and $L = 1$. For arbitrary L and unit-time tasks, they proposed an $O(n^3)$ algorithm. In what follows, we improve this result by providing an $O(n^2)$ algorithm for arbitrary L and arbitrary processing times based on dynamic programming. For the multiprocessor case we also improve the complexity from $O(n^3m^2)$ [7] to $O(n^2m)$ in the case of unit-time tasks and $L = 1$. In Table 1, we summarize the results concerning our problem (existing and new).

2. Properties for the single processor case

In the rest of the paper we assume that the instance is *agreeable*, i.e. for any pair of jobs i and j, one has $r_i \leq r_j$ if and only if $d_i \leq d_j$.

Given a schedule σ, we denote by $S_\sigma(i)$ and $C_\sigma(i)$ respectively the starting time and completion time of a job i in σ. We define $gap_\sigma(i, i+1) = S_\sigma(i+1) - C_\sigma(i)$ the length of the gap between jobs i and $i+1$.

Definition 1. An EDF (earliest deadline first) schedule is a schedule in which jobs are scheduled according to the non decreasing order of their deadlines (in case of identical deadlines, the job with the smallest release date is scheduled first).

Our algorithm relies on the following well-known result in the scheduling literature, we give its proof for completeness.

Proposition 1. *There exists an optimal solution in which all tasks are scheduled according to the* EDF *order.*

Proof. Let σ be an optimal schedule, and assume that there exist two consecutive tasks i and j such that $d_i > d_j$. Notice that a gap may exist between i and j. We consider the schedule σ' obtained from σ by swapping i and j such that $S_{\sigma'}(j) = S_\sigma(i)$ and $C_{\sigma'}(i) = C_\sigma(j)$. Since the instance is agreeable, we know that $r_i \geq r_j$. Therefore, $S_{\sigma'}(j) = S_\sigma(i) \geq r_i \geq r_j$ and $C_{\sigma'}(i) = C_\sigma(j) \leq d_j < d_i$, and the schedule σ' is feasible. The schedule σ' has the same gaps' length as σ and is therefore also optimal. By performing such swaps, one obtains an optimal schedule satisfying the EDF order. □

In the rest of the paper, we assume that the jobs $1, 2, \ldots, n$ are sorted according to the EDF order, i.e. $d_1 \leq d_2 \leq \cdots \leq d_n$ and $r_1 \leq r_2 \leq \cdots \leq r_n$.

Proposition 2. *Without loss of generality, we may assume that* $d_i \leq d_{i+1} - p_{i+1}$, *for every* $i = 1, \ldots, n-1$, *and* $r_i \geq r_{i-1} + p_{i-1}$, *for every* $i = 2, \ldots, n$.

Proof. If this is not the case, we can modify the release dates and deadlines of the tasks without modifying the optimal solution. Indeed, we can update the release dates assuming that each task is scheduled as soon as possible using the order of Proposition 1. In the same way, we can update the deadlines of the tasks, assuming that the tasks are scheduled as late as possible while keeping the order of Proposition 1. These new release dates and deadlines can be computed in the following way: let $r_i^* = \max\{r_{i-1}^* + p_{i-1}, r_i\}$, for $i = 2$ to n and $d_i^* = \min\{d_{i+1}^* - p_{i+1}, d_i\}$, for $i = n-1$ down to 1, with $r_1^* = r_1$ and $d_n^* = d_n$. □

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023

Remark 1. It is easy to see that if there exists a task i with $r_i^* + p_i > d_i^*$, where r_i^* (resp. d_i^*) is the new release date (resp. deadline) of task i, then the instance does not admit any feasible solution.

Lemma 1. *Let tasks $i, i+1, \ldots, j$ be one maximal block of continuous tasks in a schedule σ, i.e. $gap_\sigma(i, i+1) = \cdots = gap_\sigma(j-1, j) = 0$ and $gap_\sigma(j, j+1) > 0$. Then either $C_\sigma(i) = d_i$ or $C_\sigma(k) < d_k$ for $i \le k \le j$.*

Proof. The proof is by contradiction. Let us assume that $C_\sigma(i) < d_i$ and let k be the smallest index k, with $i+1 \le k \le j$, such that $C_\sigma(k) = d_k$. We have $C_\sigma(k-1) < d_{k-1}$. Therefore, $d_{k-1} > C_\sigma(k-1) = C_\sigma(k) - p_k = d_k - p_k$. This is in contradiction with the inequality $d_{k-1} \le d_k - p_k$ which comes from Proposition 2. □

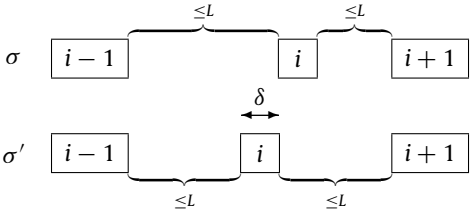
We denote by $E(\sigma)$ the energy spent by the schedule σ.

Proposition 3. *There exists an optimal solution in which all tasks are scheduled according to the* EDF *order, and such that for any task i, either task i is scheduled at*

(P1) *its release date r_i, or*
(P2) *the completion time of task $i-1$, or*
(P3) *time $d_i - p_i$.*

Proof. Let us consider an optimal EDF schedule σ which does not satisfy this proposition, and let i be the first task in this schedule which does not satisfy P1 or P2 or P3. We show how to transform this schedule to get an optimal schedule σ' in which tasks up to i satisfy P1, P2 or P3. By repeating such transformations we get an optimal schedule which satisfies Proposition 3. There are several cases to consider.

Case 1: $2 \le i \le n-1$ and $0 < gap_\sigma(i-1, i) \le L$ and $0 \le gap_\sigma(i, i+1) \le L$.



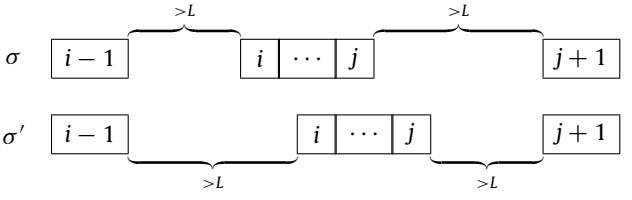
Let σ' be the schedule obtained from σ by pushing to the left the task i, i.e. by moving i to its earliest possible starting time. Let $\delta = S_\sigma(i) - S_{\sigma'}(i)$. Since σ' cannot be better than the optimal schedule σ, we have that $gap_{\sigma'}(i, i+1) \le L$, i.e. $gap_\sigma(i, i+1) + \delta \le L$. From $gap_\sigma(i-1, i) \le L$ and $gap_\sigma(i, i+1) + \delta \le L$, we can conclude that $E(\sigma') = E(\sigma)$, and therefore σ' is an optimal schedule for which task i satisfies properties P1 or P2.

Case 2: $2 \le i \le n-1$, $gap_\sigma(i-1, i) > L$ and $0 < gap_\sigma(i, i+1) \le L$.
This case cannot occur, otherwise by pushing the task i to the right we would obtain a schedule σ' with $E(\sigma') < E(\sigma)$.

Case 3: $2 \le i \le n-1$, $0 < gap_\sigma(i-1, i) \le L$ and $gap_\sigma(i, i+1) > L$.
This case cannot occur, otherwise by pushing the task i to the left we would obtain a schedule σ' with $E(\sigma') < E(\sigma)$.

Case 4: $2 \le i \le n-1$, $gap_\sigma(i-1, i) > L$ and $gap_\sigma(i, i+1) > L$.
Let σ' be the schedule obtained from σ by pushing to the left the task i, i.e. by moving i to its earliest possible starting time. Let $\delta = S_\sigma(i) - S_{\sigma'}(i)$. Since σ' cannot be better than the optimal schedule σ, we have that $gap_{\sigma'}(i-1, i) > L$, i.e. $gap_\sigma(i-1, i) > L + \delta$. From $gap_\sigma(i-1, i) > L + \delta$ and $gap_\sigma(i, i+1) > L$, we can conclude that $E(\sigma') = E(\sigma)$, and therefore σ' is an optimal schedule for which task i satisfies properties P1 or P2.

Case 5: $2 \le i \le n-1$, $gap_\sigma(i-1, i) > L$ and $gap_\sigma(i, i+1) = 0$.



Let $j \ge i+1$ be the largest index such that $gap(k-1, k) = 0$ for $i+1 \le k \le j$. Tasks $i, i+1, \ldots, j$ form a maximal block of continuous tasks. To obtain the schedule σ' we push this block to the rightmost extent. Let $\delta = S_{\sigma'}(i) - S_\sigma(i) = \cdots = S_{\sigma'}(j) - S_\sigma(j)$ the shift amount. Since task i does not satisfy property P3, $C_\sigma(i) \ne d_i$, and hence from Lemma 1 we get that $C_\sigma(k) < d_k$ for $i \le k \le j$, and therefore $\delta > 0$. Since in schedule σ' the block of tasks is pushed to its rightmost extent, we can conclude from Lemma 1 that $C_{\sigma'}(i) = d_i$, and so task i satisfies property P3 in the schedule σ'. Notice that since $gap_\sigma(i-1, i) > L$, one must have $gap_{\sigma'}(j, j+1) \ge L$, otherwise the schedule σ' would be better than σ. From

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023

$gap_\sigma(i-1,i) > L$ and $gap_\sigma(j,j+1) \geq L+\delta$, we can conclude that $E(\sigma') = E(\sigma)$, and therefore σ' is an optimal schedule for which task i satisfies P3.

Case 6: $i = 1$ and $gap_\sigma(i,i+1) = 0$.
This case is similar to Case 5.

Case 7: $i = 1$ and $0 < gap_\sigma(i,i+1) \leq L$.
This case cannot occur, otherwise by pushing to the right the task i we would obtain a schedule σ' with $E(\sigma') < E(\sigma)$.

Case 8: $i = 1$ and $gap_\sigma(i,i+1) > L$.
By pushing the task i to the right, we obtain a schedule σ' with $E(\sigma') = E(\sigma)$, such that i satisfies P3.

Case 9: $i = n$ and $0 < gap_\sigma(i-1,i) \leq L$.
This case cannot occur, otherwise by pushing to the left the task i we would obtain a schedule σ' with $E(\sigma') < E(\sigma)$.

Case 10: $i = n$ and $gap_\sigma(i-1,i) > L$.
By pushing the task i to the left, we obtain a schedule σ' with $E(\sigma') = E(\sigma)$, such that i satisfies P1 or P2. □

Proposition 4. *There are at most k possible positions for task k.*

Proof. Let us denote by H_k the induction hypothesis, i.e. that there are at most k positions for task k.

Base case: The first task has only one position and $S_\sigma(1) = d_1 - p_1$. H_1 is true. Note that the first task does not verify P1, because the energy cannot be better in this case.

Inductive step: Let us assume H_k, and prove H_{k+1}.

We have at most k positions for task k. According to Proposition 3, task $k+1$ can be scheduled at its release date, at its deadline, or after all the positions of task k. Then, task $k+1$ has at most $k+2$ possible positions. However, if tasks k and $k+1$ are scheduled at their release dates:

- either $gap_\sigma(k,k+1) = 0$, then task $k+1$ is scheduled immediately after task k, and it is the same position as the release date of task $k+1$. Then there are at most $k+1$ positions for task $k+1$,
- either $gap_\sigma(k,k+1) > 0$, then task $k+1$ cannot be scheduled immediately after task k, and there are also at most $k+1$ positions for task $k+1$.

Hence, H_{k+1} is true. □

3. A dynamic program

We use the following notations:

- D_k: the set of possible completion times of the kth task,
- $E_k(t)$: the energy spent during idle periods in an optimal subschedule σ over the k first tasks with the constraint that $C_\sigma(k) = t \in D_k$.

According to Proposition 1, we can process every task in the EDF order.

One has $D_1 = \{d_1\}$, and the sets D_k can be computed according to Proposition 3 in the following way: $D_k = \bigcup_{t \in D_{k-1}} \{t + p_k | t \geq r_k\} \cup \{r_k + p_k\} \cup \{d_k\}$.

For a fixed task k, we look for the best subschedule of the $k-1$ first tasks by considering the energy spent during the corresponding idle periods and the new gap between tasks $k-1$ and k.

If task k is scheduled at $S_\sigma(k) = r_k$ or $S_\sigma(k) = d_k - p_k$, then we look for all possible positions of task $k-1$ scheduled before task k. Otherwise, according to Proposition 3, there is no gap between tasks $k-1$ and k.

The energy of an optimal schedule is given by $\min_{t \in D_n} E_n(t)$.

One has $E_1(d_1) = 0$, and

$$\forall t \in D_k, \quad E_k(t) = \begin{cases} \min\limits_{t' \in D_{k-1}} \{E_{k-1}(t') + \Delta(t', t - p_k)\} & \text{if } t = d_k \text{ or } t = r_k + p_k \\ E_{k-1}(t - p_k) & \text{otherwise} \end{cases}$$

with

$$\Delta(t', t) = \begin{cases} \min\{L, t - t'\} & \text{if } t - t' \geq 0 \\ +\infty & \text{otherwise.} \end{cases}$$

The function Δ returns the energy spent during the idle period between two dates t' and t. If $t' > t$, then task $k-1$ is scheduled after task k, thus the function returns $+\infty$ since the schedule is not feasible.

Proposition 5. *The complexity of the dynamic program is $O(n^2)$.*

Proof. For a fixed k, the set of values $E_k(t)$ for $t \in D_k$ can be computed in time $O(k)$. Indeed, according to Proposition 4, one has $|D_k| \leq k$. The two values $E_k(t)$ when $t = d_k$ or $t = r_k + p_k$ are computed in time $O(k-1)$ since $|D_{k-1}| \leq k-1$. Each of the $|D_k| - 2$ remaining values can be computed in time $O(1)$. Finally, the problem can be solved in time $\sum_{k=1}^{n} O(k) = O(n^2)$. □

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023

4. Properties for the multiprocessor case

In the following, we focus on the multiprocessor case with unit-time tasks and $L = 1$. We count an additional cost if a task is scheduled on a new processor.

Definition 2. A schedule is called compact if whenever a job j is scheduled at time t on a processor $p > 1$, then all lower-numbered processors $1 \leq q < p$ are also occupied at time t.

Assume that t is the first instant at which the schedule is no more compact. Let q be the processor which is idle at time t and p another processor such that $p < q$ executing some task at the same time. Then by switching all the tasks executed on p and q after t, the number of gaps is not increased. This operation can be repeated until the schedule becomes compact.

Proposition 6 ([6])**.** *Any feasible instance of multiprocessor gap scheduling has an optimal solution which is a compact schedule.*

Because of Proposition 6, all the considered schedules in the rest of the paper are compact.

Proposition 7. *There exists an optimal compact schedule such that all tasks are scheduled following the* EDF *order. If two tasks are scheduled at the same time but on different processors, then these tasks are scheduled according to the* EDF *order by considering the machines in increasing order.*

Proof. The first part comes from the proof of Proposition 1. Now let us assume that σ is an optimal compact schedule in which the tasks are scheduled according to EDF order, i.e. $\forall i < j, S_\sigma(i) \leq S_\sigma(j)$. For every pair of tasks $i < j$ such that $S_\sigma(i) = S_\sigma(j)$, denote by Π_i the processor which executes the task i, and by Π_j the processor which executes the task j. If $\Pi_i > \Pi_j$, tasks i and j can be swapped without changing the optimality. Hence, there exists an optimal compact schedule such that for a fixed time, tasks are ordered according to the EDF by considering the machines in increasing order. □

Proposition 8. *Without loss of generality, we may assume that at most m tasks have the same release date, and at most m tasks have the same deadline.*

Proof. If this is not the case, we can modify the release dates and deadlines of the tasks without modifying the optimal solution. Indeed, we can update the release dates assuming that each task is scheduled as soon as possible using the order of Proposition 7. In the same way, we can update the deadlines of the tasks, assuming that the tasks are scheduled as late as possible while keeping the order of Proposition 7. These new release dates and deadlines can be computed in the following way:
For i from 1 to m, $r_i^* := r_i$
For i from $m+1$ to n, if $r_i = r_{i-m}^*$, then $r_i^* := r_i + 1$, otherwise $r_i^* := r_i$
For i from n to $n-m+1$, $d_i^* := d_i$
For i from $n-m$ to 1, if $d_i = d_{i+m}^*$, then $d_i^* := d_i - 1$, otherwise $d_i^* := d_i$. □

Remark 2. As in Proposition 2, we can modify the release dates and the deadlines such that they verify Proposition 8. Then it is easy to verify if an instance admits a feasible solution.

Definition 3. An interval $I^* = [s, t]$ is called *critical* if $|\{1 \leq i \leq n : [r_i, d_i] \subseteq [s, t]\}| > t - s$, in other words it is an interval for which there are more tasks to schedule than its length. A task is called critical if this task is in at least one critical interval.

The following lemma can be easily proved.

Lemma 2. *A task k is critical iff there exist $1 \leq i < j \leq n$, with $i \leq k \leq j$, such that $j - i + 1 > d_j - r_i$.*

Definition 4. Given a schedule σ, a set of tasks $i, i+1, \ldots, j$ is called a maximal block of tasks if $C_\sigma(k) = (S_\sigma(k+1)$ or $C_\sigma(k+1))$ for all $i \leq k < j$, and $gap_\sigma(j, j+1) > 0$ and $gap_\sigma(i-1, i) > 0$. If all the tasks are scheduled on the first processor, we say it is a low block of tasks. If there exists at least one task which is scheduled on the second processor, we say it is a high block of tasks.

Let Π_k be the processor on which task k is executed. Let us consider an optimal schedule σ, and let $i, \ldots, k, \ldots, k', \ldots, j$ be a maximal high block of tasks such that:

- k is the first task in the block with $\Pi_k = 1$ and $\Pi_{k+1} = 2$,
- k' is the last task with $\Pi_{k'} > 1$.

We define two transformations on σ such that the new obtained schedule σ' is also optimal:

- **Transformation A** (see Fig. 1): If $C_\sigma(k') \neq d_{k'}, C_\sigma(k'+1) \neq d_{k'+1}, \ldots, C_\sigma(j) \neq d_j$, then we push to the right the last task of the block, i.e. task k' is scheduled instead of $k'+1$, $k'+1$ instead of $k'+2$, and so on. One obtains $C_{\sigma'}(h-1) = C_\sigma(h)$ for h from $k'+1$ to j, and $C_{\sigma'}(j) = C_\sigma(j) + 1$. Notice that if $C_{\sigma'}(j) = S_{\sigma'}(j+1)$, we obtain a new maximal block, on which it may be possible to apply Transformations A or B again.

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023

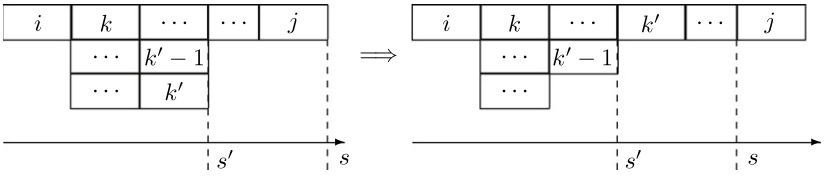


Fig. 1. Illustration of Transformation A.

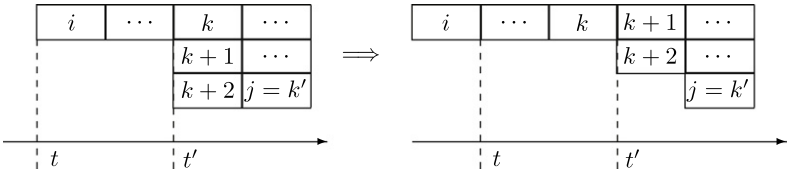


Fig. 2. Illustration of Transformation B.

- **Transformation B** (see Fig. 2): If $S_\sigma(i) \neq r_i, S_\sigma(i+1) \neq r_{i+1}, \ldots, S_\sigma(k) \neq r_k$, then we push to the left the first task of the block, i.e. task k is scheduled instead of $k-1$, $k-1$ instead of $k-2$, and so on. Moreover, tasks that are scheduled at the same time as task k have to be scheduled at the same time, but in a lower-numbered processor in order to get a compact schedule. One obtains $C_{\sigma'}(h+1) = C_\sigma(h)$ for h from i to $k-1$, and $C_{\sigma'}(i) = C_\sigma(i) - 1$. Note that if $S_{\sigma'}(i) = C_{\sigma'}(i-1)$, we obtain a new maximal block, on which it may be possible to apply Transformations A or B again.

  Notice that Transformations A and B are not defined for a low block of tasks.

Proposition 9. *If σ is an optimal schedule, then the schedules obtained from it by applying Transformations A and B are also optimal.*

Proof. We prove the proposition only for Transformation A since it is similar for Transformation B. Let us denote $p = \Pi_{k'}$ in σ. Let us also denote $[s, t]$ the idle period on the first processor after the block and $[s', t']$ the idle period on the processor p after the same block. We can assume that $s' \leq s \leq t \leq t'$ according to Proposition 6. Suppose now that we change the schedule using Transformation A. Without considering the tasks, we can see that the gaps become respectively $[s+1, t]$ for the first processor, and $[s'-1, t']$ for the processor p or $[r, t']$ (with $r < s'$) if there is no other task in the block on processor p. The energy consumed in the first gap decreases by one if $t - s \leq L$, and it does not change otherwise. The energy consumption in the second gap increases by one if $t' - s' < L$, but it does not change otherwise. Since $s' \leq s \leq t \leq t'$, we cannot have $t' - s' < L$ and $t - s > L$. Thus, the energy consumption does not increase. □

Lemma 3. *There exists an optimal solution such that all the non critical tasks are scheduled on the first processor.*

Proof. Let us consider an optimal schedule σ such that we cannot apply Transformations A or B anymore (by Proposition 9 such an optimal schedule exists). Let us consider a task t which is not scheduled on the first processor. We show that this task is necessarily critical. This task belongs to a high block of tasks in σ. Since we cannot apply Transformations A or B anymore, it means that there exist tasks k and k' in the block such that: $k < k'$, $C_\sigma(k) = r_k + 1$ and $C_\sigma(k') = d_{k'}$. Moreover one has $k + 1 \leq t \leq k'$, since by the way Transformations A and B are designed, the tasks i within the block such that $i \leq k$, or $i \geq k' + 1$ (if such tasks exist), are necessarily scheduled on the first processor. We can conclude that tasks $k, \ldots, k'$ are critical since $[r_k, d_{k'}]$ is a critical interval. Indeed one has $k' - k + 1 > d_{k'} - r_k$, since even by scheduling task k (resp. k') as soon (resp. late) as possible, it is not possible in the compact schedule σ (in which there is no idle time in the block on the first processor) to schedule all the tasks from k to k' on the first processor. □

Proposition 10. *There exists an optimal solution in which all tasks are scheduled according to the* EDF *order, and such that for any non critical task i, either task i is scheduled at*

(P1) *its release date r_i, or*
(P2) *the completion time of task $i-1$: $S_\sigma(i) = C_\sigma(i-1)$, or*
(P3) *time $d_i - 1$, or*
(P4) *before the starting time of task $i+1$: $C_\sigma(i) = S_\sigma(i+1)$,*

and such that for each maximal block of tasks, there exists a task which verifies P1 *or* P3*. Moreover, task i is scheduled on the first processor. Such a solution will be called a canonical optimal schedule in the rest of the paper.*

Proof. Let us consider an optimal schedule σ which satisfies Lemma 3. We construct a canonical schedule σ' from σ in two phases. In the first phase, while there exists a high block of tasks on which we can apply Transformations A or B, we do it. Then, in the second phase we consider the low blocks of tasks and we move those blocks as described in the proof of Proposition 3.

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023

In the following cases, we consider a maximal block of tasks in σ'.

Case 1: The block is a low block.

This block may contain critical and non critical tasks. Since in the second phase, we moved this block as described in the proof of Proposition 3, so the Proposition is satisfied.

Case 2.1: The block is a high block, and all its tasks are non critical.

This case cannot occur because by using Lemma 3 we started from a schedule in which all the non critical tasks were scheduled on the first processor.

Case 2.2: The block is a high block, and there are some critical and non critical tasks.

Let us denote by i (resp. j) the first (resp. last) task in the block, i.e. the task with the smallest (resp. greatest) index. Since we cannot apply Transformations A and B on this block, it means that there exists at least one task which satisfies P1, and at least one task which satisfies P3 (it can be the same task). Let us denote by i_1 (resp. i_2) the first (resp. last) task which satisfies P1 (resp. P3).

Clearly we cannot have $i_2 < i_1$, and tasks $i, \ldots, i_1$ are scheduled on the first processor, and tasks $i_2 + 1, \ldots, j$ are also scheduled on the first processor. Then, tasks $i_1, \ldots, i_2$ are necessarily critical since $i_2 - i_1 + 1 > d_{i_2} - r_{i_1}$.

Case 2.3: The block is a high block, and all its tasks are critical.

Since we cannot apply Transformations A and B on it, it means that there exists a task which verifies P1 or P3. □

In the following we show that there are a few relevant time points at which tasks end in some optimal schedule. Let $\mathcal{I} = \{l_1, l_2, \ldots, l_{|\mathcal{I}|}\}$ be the set of non critical tasks. We compute the set D_k for each task k in the following way.

Step 1. If k is a critical task, then $D_k := \{r_k + 1, r_k + 2, \ldots, d_k\}$, otherwise $D_k := \emptyset$, for k from 1 to n.

Step 2. $D_{l_1} := \{r_{l_1} + 1\} \cup \{d_{l_1}\}$.

Step 3. $D_{l_k} := D_{l_k} \cup \bigcup_{t \in D_{l_k - 1}} \{t + 1 | r_{l_k} < t + 1 \leq d_{l_k}\} \cup \{r_{l_k} + 1\} \cup \{d_{l_k}\}$ for k from 2 to $|\mathcal{I}|$.

Step 4. $D_{l_k} := D_{l_k} \cup \bigcup_{t \in D_{l_k + 1}} \{t - 1 | r_{l_k} < t - 1 \leq d_{l_k}\}$ for k from $|\mathcal{I}| - 1$ down to 1.

Notice that applying Step 3 again after the end of Step 4 would not change the sets D_k.

Proposition 11. *There exists an optimal schedule σ such that for every task k, $C_\sigma(k) \in D_k$.*

Proof. Let σ be a canonical optimal schedule according to Proposition 10 such that we cannot apply Transformations A or B anymore. We want to show that $C_\sigma(k) \in D_k$ for every task k. The proof is by contradiction. Let us assume that there exists a task k such that $C_\sigma(k) \notin D_k$. Notice that k is necessarily a non critical task, and is therefore scheduled on the first processor according to Proposition 10. We consider the maximal block of tasks $i, \ldots, k, \ldots, j$ in which k belongs.

Case 1: It is a low block.

Since $C_\sigma(k) \notin D_k$, we have $C_\sigma(k-1) \notin D_{k-1}$ and $C_\sigma(k+1) \notin D_{k+1}$, and so on. Indeed, observe for example that if $C_\sigma(k-1) \in D_{k-1}$, then by the way D_k is computed it would imply that $C_\sigma(k) \in D_k$.

Then, for each task l of this block, $i \leq l \leq j$, one has $C_\sigma(l) \notin D_l$, and therefore $C_\sigma(l) \notin \{r_l + 1, d_l\}$. Then no task in the block verifies P1 or P3, contradicting Proposition 10.

Case 2: It is a high block.

According to Proposition 10, the tasks which are not scheduled on the first processor are necessarily critical. Let us denote by i_1 the first task such that task $i_1 + 1$ is scheduled on the second processor, and i_2 the last task which is not scheduled on the first processor.

Tasks $i_1, \ldots, i_2$ are critical and task k cannot be one of $i_1, \ldots, i_2$. Suppose that $i \leq k < i_1$. We have $C_\sigma(k) \notin D_k$ and $C_\sigma(k+1) \notin D_{k+1}$ and so on until a critical task t. We have a contradiction, since for a critical task, its completion time is always included in D_t. The proof is similar in case $i_2 < k \leq j$. □

Proposition 12. *The size of D_k is $O(n)$ for each task k.*

Proof. If k is a critical task, it belongs to a critical interval $I = [s, t]$, with $s \leq r_k < d_k \leq t$. Since I is critical, it means that $d_k - r_k \leq t - s < |\{1 \leq i \leq n : [r_i, d_i] \subseteq [s, t]\}| \leq n$. Therefore, $|D_k| = |\{r_k + 1, r_k + 2, \ldots, d_k\}| < n$.

When we apply Step 2 and Step 3, there are still at most $O(n)$ positions for each task. This is still the case when we apply Step 4. □

Proposition 13. *The sets D_k, for $1 \leq k \leq n$, can be computed in time $O(n^2)$.*

Proof. In Step 1 we need to compute the set of critical tasks. This can be done in time $O(n^2)$. From Lemma 2 we get that, for $1 \leq i < j \leq n$, if $j - i > d_j - r_i$, then tasks $i, \ldots, j$ are critical, and all critical tasks can be found in this way. By Proposition 12 the size of each set D_k is $O(n)$ and therefore the computation time of Step 2 to Step 4 is $O(n)$. □

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023

5. A dynamic program for the multiprocessor case with $L = 1$

In this section, we show that it is possible to use the properties derived in the previous section in order to obtain an efficient dynamic programming algorithm that returns an optimal solution.

Recall that D_k is the set of possible completion times of the kth task, computed in the previous section.

Definition 5. A (k, t, p)-cos (for constrained optimal schedule) is an optimal (sub)schedule σ over tasks $1, 2, \ldots, k$, under the constraints that task k is scheduled at time $t-1$ on processor p, and such that $C_\sigma(i) \in D_i$ for all tasks $1 \leq i \leq k$.

Given a schedule σ, we denote by $s_\sigma(t)$ the largest processor on which there is a task executed in the time interval $[t-1, t]$. If there is no task executed in the time interval $[t-1, t]$, then $s_\sigma(t) = 0$. The profile of σ, denoted by $P(\sigma)$, is the vector $(s_{C_\sigma(n)}, s_{C_\sigma(n)-1}, \ldots, s_1)$.

Given two vectors $u = (u_1, \ldots, u_\alpha)$ and $v = (v_1, \ldots, u_\beta)$, we say that u is lex-greater than v if either

1. $\alpha > \beta$ or
2. $\alpha = \beta$ and u is lexicographically greater than v.

Given two schedules σ and σ', we say that σ is lex-greater than σ' if $P(\sigma)$ is lex-greater than $P(\sigma')$.

Definition 6. The canonical (k, t, p)-cos is defined as the (unique) schedule σ such that σ is a (k, t, p)-cos and there does not exist a (k, t, p)-cos σ' such that σ' is lex-greater than σ.

We denote by $E_k(t, p)$ the energy spent during idle periods and the energy spent for the transition from the sleep to the active state in the canonical (k, t, p)-cos, and $F_k(t, q)$ the completion time of the processor q in the canonical $(k, t, 1)$-cos. The function $\Delta(t', t)$ is the same as the one in the first algorithm.

Notice that the energy of an optimal schedule is given by
$\min_{t \in D_n, 1 \leq p \leq m} E_n(t, p)$.

We now consider how to compute the values $E_k(t, p)$. For the first task, one has $E_1(t, 1) := 1, \forall t \in D_1$. Since we consider only compact schedules, the first task must be scheduled on processor $p = 1$. Therefore, one has $F_1(t, 1) := t, \forall t \in D_1$ and $F_1(t, q) := -\infty, \forall q \neq 1, \forall t \in D_1$.

In each step, $E_k(t, p)$ is computed from previous values $E_{k-1}(t', p')$.

There are several cases to consider:

Case 1: If $p = 1$,

- **Case 1.1**: If $t - 1 \notin D_{k-1}$ and $D_{k-1} \cap]-\infty, t-2] := \emptyset$,

  $E_k(t, p) = +\infty$.

- **Case 1.2**: If $t - 1 \notin D_{k-1}$ and $D_{k-1} \cap]-\infty, t-2] \neq \emptyset$,

  $E_k(t, p) := E_{k-1}(t^*, q^*) + \Delta(t^*, t-1)$ with

  $$(t^*, q^*) := \max_q \left\{ \max_{t'} \left\{ \operatorname*{argmin}_{t' \in D_{k-1}, t' < t, 1 \leq q \leq m} \{E_{k-1}(t', q) + \Delta(t', t-1)\} \right\} \right\}.$$

  We update the completion times on each processor in the following way:

  $F_k(t, p) := t$,
  $F_k(t, p') := F_{k-q^*}(t^*, p')$, for p' from 2 to m.

- **Case 1.3**: If $t - 1 \in D_{k-1}$
  $E_k(t, p) := E_{k-1}(t-1, q^*)$ with $q^* := \max_q \{\operatorname*{argmin}_{1 \leq q \leq m} \{E_{k-1}(t-1, q)\}\}$.
  We update the completion times on each processor in the following way:
  $F_k(t, p) := t$,
  $F_k(t, p') := t - 1$, for p' from 2 to q^*,
  $F_k(t, p') := F_{k-q}(t-1, p')$, for p' from $q^* + 1$ to m.

Case 2: If $p > 1$,

- **Case 2.1**: If $E_{k-1}(t, p-1) = +\infty$ or $t > d_{k-p+1}$,

  $E_k(t, p) := +\infty$.

- **Case 2.2**: Task k is a non critical task.

  $E_k(t, p) := +\infty$.

- **Case 2.3**: Otherwise,

  $E_k(t, p) := E_{k-1}(t, p-1) + \Delta(F_{k-p+1}(t, p), t-1)$.

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023

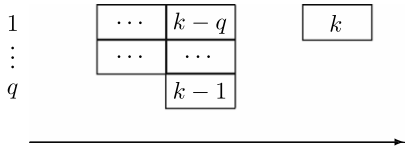


Fig. 3. Illustration of Case 1.2: if $\Pi_k = 1$ with $C_\sigma(k) = t$ and $C_\sigma(k-1) < t-1$, then task $k-q$ is scheduled on processor 1 at time $C_\sigma(k-1)$.

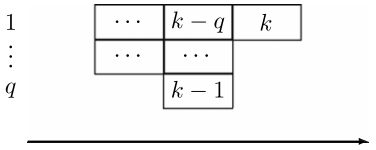


Fig. 4. Illustration of Case 1.3: if $\Pi_k = 1$ with $C_\sigma(k) = t$ and $\Pi_{k-1} = q$ with $C_\sigma(k-1) = t-1$, then $\Pi_{k-q} = 1$ with $C_\sigma(k-q) = t-1$.

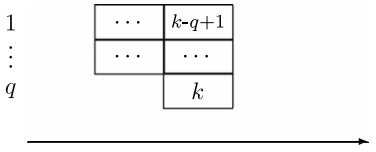


Fig. 5. Illustration of Case 2.3: if $\Pi_k = q$, then task $k-q+1$ is scheduled on processor 1 for a given time.

In Case 1.2, (t^*, q^*) is obtained by computing first the greatest value t', and then the greatest value q such that $E_{k-1}(t', q) + \Delta(t', t-1)$ is minimized.

Proposition 14. *The dynamic programming correctly computes $E_k(t, p)$ for all $k = 1, \ldots, n$, $t \in D_k$, $p = 1, \ldots, m$.*

Proof. Let us consider the notion of *front*, which is defined as the "height" of the last task in a schedule: if the last task is scheduled on processor p, then the front has a height equal to p. A useful remark is that if several schedules have the same energy spent during idle periods with the same completion time for the last task, then it is always better to keep the schedule with the largest front. Indeed, let us consider two (k, t, p)-cos and (k, t, p')-cos with the same energy spent, such that $p' < p$. If task $k+1$ cannot be scheduled at $C_{(k,t,p)-cos}(k)$, then there is a gap, and whatever the sub-schedule we choose, there is no change. Whereas, if task $k+1$ can be scheduled at $C_{(k,t,p)-cos}(k)$, then there is no gap. If the next tasks $k+2$ must be scheduled at $C_{(k,t,p)-cos}(k)$, there will be a smaller gap if we choose the (k, t, p)-cos instead of the (k, t, p')-cos. Thus, each time it is possible that we select the (k, t, p)-cos with the largest p possible without deteriorating the solution.

We now consider several cases:

Cases 1.1 and 2.1 represent infeasible cases.

Case 1.2: In this case, depicted in Fig. 3, there must be a gap between tasks $k-1$ and k. In this case, if σ is a canonical $(k, t, 1)$-cos, then the subschedule up to task $k-1$ is a canonical $(k-1, t^*, q^*)$-cos with t^* and q^* computed as above.

Without loss of generality, in order to obtain a low complexity algorithm, this case is computed once for a fixed task k for all $t \in D_k$ with $t > d_{k-1}$.

Case 1.3: In this case, depicted in Fig. 4, if σ is a canonical $(k, t, 1)$-cos, then the subschedule up to task $k-1$ is a canonical $(k-1, t-1, q^*)$-cos with q^* computed as above. Indeed, let us consider σ, a canonical $(k, t, 1)$-cos such that there exists a gap between tasks $k-1$ and k. We can transform σ into σ', in the following way: we change the execution time of task $k-1$ such that $C_{\sigma'}(k-1) = S_{\sigma'}(k)$. It is easy to see that by moving task $k-1$, the gap described in σ is the same as the gap before task $k-1$ in σ', and σ would not be a canonical $(k, t, 1)$-cos. Thus, to compute $E_k(t, 1)$, we can only consider task $k-1$ which is scheduled immediately before task k. According to the remark above, we retain the schedule with the biggest front.

Case 2.2: This case comes from Lemma 3.

Case 2.3: This case is depicted in Fig. 5. Since $E_{k-p+1}(t, p)$ is computed by selecting the sub schedule with the biggest front in Case 1.3, then without loss of generality, a (k, t, p)-cos is computed from a $(k-1, t, p-1)$-cos and is optimal. The completion time can be found when task $k-p+1$ is scheduled on the first processor since the $(k-1, t, p-1)$-cos is computed from a $(k-2, t, p-2)$-cos, and so on, until the first processor.

Feasibility: There exists at least one (k, t, p)-cos with $E_k(t, p) < +\infty$ for each task k.

We suppose that the instance admits a feasible schedule according to Remark 2.

Suppose that task k is non critical. Then there exists an $E_k(t, 1)$, $\forall t \in D_k$ which is finite. If it is not the case, then task k is a critical task and we have a contradiction. Suppose now that task k is critical. Then for a fixed time t, there exists at least a processor p where task k is scheduled. Suppose that task k is scheduled at its release date. Then, there exists a $(k, r_k + 1, p)$-cos for which $E_k(r_k + 1, p)$ is finite. If it is not the case, then we have a contradiction with Proposition 8. □

Proposition 15. *The dynamic program has a complexity of $O(n^2 m)$.*

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023

Proof. There are $O(n^2m)$ values $E_k(t, p)$ and $O(n^2m)$ values $F_k(t, q)$ to be computed. First, we analyze each case for a fixed task k. Cases 1.1 and 2.1 are infeasible and can occur $O(n^2m)$ times.
Case 1.2: For fixed t, $E_k(t, 1)$ is computed in time $O(m)$ since $C_\sigma(k-1) = S_\sigma(k)$ and there are $O(m)$ values to consider. Each value of $F_k(t, q)$ is computed in time $O(1)$, and there are $O(m)$ values to be computed. This case occurs $O(n)$ times according to Proposition 12. Thus, Case 1.2 is computed in time $O(nm)$.
Case 1.3: $E_k(t, 1)$ is computed in time $O(nm)$. Since it is the same solution whatever the position of task k, we can compute this value once, and use it each time this case occurs. For fixed t, $F_k(t, q)$ is computed in time $O(m)$. Similarly to the Case 1.2, this case occurs $O(n)$ times. Thus, Case 1.3 is computed in time $O(nm)$.
Case 2.2: For fixed t, p, $E_k(t, p)$ is computed in time $O(1)$. This case can occur $O(nm)$ times.
Case 2.3: For fixed t, p, $E_k(t, p)$ is computed in time $O(1)$. This case occurs $O(nm)$ times for all values of t and $p \neq 1$. Thus, Case 2.3 is computed in time $O(nm)$.

Finally, for fixed task k, $E_k(t, p)$ is computed in time $O(nm) + O(nm) + O(nm) = O(nm)$. Then, the global problem is solved in time $O(n^2m)$. □

References

[1] S. Albers, Energy-efficient algorithms, Commun. ACM 53 (5) (2010) 86–96.
[2] P. Baptiste, Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management, in: Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 364–367.
[3] P. Baptiste, M. Chrobak, Ch. Dürr, Polynomial time algorithms for minimum energy scheduling, in: Proc. 15th Annual European Symposium of Algorithms, in: LNCS, vol. 4698, 2007, pp. 136–150.
[4] P. Baptiste, E. Néron, F. Sourd, Modèles et Algorithmes en Ordonnancement, Ellipses Edition, 2004, pp. 198–203.
[5] P. Chrétienne, On the no-wait single-machine scheduling problem, in: Proc. 7th Workshop on Models and Algorithms for Planning and Scheduling Problems, 2005.
[6] E.D. Demaine, M. Ghodsi, M.T. Hajiaghayi, A.S. Sayedi-Roshkhar, M. Zadimoghaddam, Scheduling to Minimize Gaps and Power Consumption, SPAA, 2007, pp. 46–54.
[7] Gururaj, Jalan, Stein, unpublished work, see survey of M. Chrobak, http://www.cs.pitt.edu/~kirk/cs3150spring2010/10071.ChrobakMarek.Slides.pdf, 2010.
[8] S. Irani, K.R. Pruhs, Algorithmic problems in power management, SIGACT News 36 (2) (2005) 63–76.

Please cite this article in press as: E. Angel, et al., Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines, Discrete Applied Mathematics (2014), http://dx.doi.org/10.1016/j.dam.2014.05.023