Dynamic Voltage Scaling under EDF Revisited

Bruno Gaujal (bruno.gaujal@imag.fr)
INRIA and ID-IMAG Laboratory, 51 avenue Jean Kuntzmann, 38330 Montbonnot - France

Nicolas Navet (nicolas.navet@loria.fr)
INRIA-LORIA, Campus Scientifique, BP-139, 54506 Vandoeuvre-lès-Nancy, France

Abstract. Basic algorithms have been proposed in the field of low-power (Yao et al., 1995) which compute the minimum energy-schedule for a set of non-recurrent tasks (or jobs) scheduled under EDF on a dynamically variable voltage processor. In this study, we propose improvements upon existing algorithms with lower average and worst-case complexities. They are based on a new EDF feasibility test that helps to identify the "critical intervals". The complexity of this feasibility test depends on structural characteristics of the set of jobs. More precisely, it depends on how tasks are included one in the other. The first step of the algorithm is to construct the Hasse diagram of the set of tasks where the partial order is defined by the inclusion relation on the tasks. Then, the algorithm constructs the shortest path in a geometrical representation at each level of the Hasse diagram. The optimal processor speed is chosen according to the maximal slope of each path.

Keywords: Real-time systems, low-power design, scheduling, complexity, dynamic voltage scaling

1. Introduction

Two important problems related to the scheduling of a set of independent jobs under EDF are addressed here : 1) the feasibility analysis and 2) the minimization of the energy consumption.

Existing solutions show that these problems are closely related; the optimal energy minimization algorithm from Yao et al., see (Yao et al., 1995), embeds the feasibility analysis from Spuri (Stankovic et al., 1998) although both results have been developed independently. The solutions of the two problems are also tightly linked with the new algorithms developed in this study.

Existing work Amongst hardware and software techniques aimed at reducing energy consumption, supply voltage reduction is particularly effective. This is because the energy consumption of the processor is a function which is at least quadratic in the voltage of the processor, see (Gruian, 2002) for more details. However, voltage reduction requires the reduction of the maximal frequency of the processor. This implies that a tradeoff has to be found between energy consumption and performances.

In the last few years, variable voltage processors have become available (e.g., implemented with technologies such as Transmeta Crusoe, Intel Speedstep and XScale, and AMD PowerNow!) and a lot of research has been conducted in the field of dynamic voltage scaling. When real-time constraints are matter of concern, the extent to which the system can reduce the CPU frequency depends on the tasks' characteristics (execution times, arrival dates, deadlines ...) and on the underlying scheduling policy.

Power-conscious versions of the two classical real-time scheduling policies, namely EDF (Earliest Deadline First) and FPP (Fixed Priority Pre-emptive), have been proposed. For FPP, Shin and Shoi (Shin and Choi, 1999) presented a simple run-time strategy that reduces energy consumption. Quan and Hu proposed a more efficient solution in (Quan and Hu, 2001) as well as an optimal solution in (Quan and Hu, 2002) having an exponential algorithmic complexity. Recently, Yun and Kim in (Yun and Kim, 2003) proved that computing the voltage schedule of a set of tasks under FPP is NP-Hard and they presented an approximation scheme that runs in polynomial time and whose gap w.r.t. the optimal solution can be chosen arbitrarily small.

When the scheduling is made using EDF, Yao et al. in (Yao et al., 1995) proposed an off-line algorithm to find the optimal voltage schedule of a set of independent jobs. Recently, an approach solving the same problem through a shortest path algorithm has been proposed in (Gaujal et al., 2003; Gaujal et al., 2005) but it is restricted to jobs with FIFO real-time constraints ($a_i \leq a_j \rightarrow d_i \leq d_j$ where a_i and d_i are the arrival time and the deadline of job i, respectively). For such FIFO jobs, feasibility testing and energy minimization can be solved

in $O(N \log(N))$ versus $O(N^3)$ with the Yao et al's algorithm (for more details on the complexity analysis, please refer to paragraph 2.3). The problem of finding the optimal voltage schedule has also been solved for recurrent tasks: periodic tasks in (Liu and Mok, 2003) and sporadic tasks in (Qadi et al., 2003; Scordino and Lipari, 2006). It is worth noting that several results, such as the uniqueness of the speed function and the characterization of the points in time at which speed changes occur, have been independently published in (Liu and Mok, 2003) and (Gaujal et al., 2003; Gaujal et al., 2005).

Other directions of research concern the discrepancy between worst-case execution times (WCET) and actual execution times. A first class of algorithms, known as "stochastic scheduling" (Lorch and Smith, 2001; Gruian, 2001; Gruian, 2002; Xu et al., 2004) consists of finding a feasible speed schedule that minimizes the expectation of energy consumption. A second class of techniques (Mossè et al., 2000; Shin et al., 2001; AbouGhazaleh et al., 2003) is referred as "compiler-assisted scheduling". A task is divided into sections for which the WCET is known and the processor speed is re-computed at the end of execution of each section according to the difference between the WCET and the time that was actually needed to execute the code. Other strategies consist in dynamically collecting at the end of execution of each task the unused computation time and share it among active tasks. Among those approaches, usually termed "dynamic reclaiming algorithms", one can cite (Aydin et al., 2001; Pillai and Shin, 2001), (Zhang and Chanson, 2002) and (Liu and Mok, 2003). Finally, some techniques, such as in (Aydin et al., 2004), anticipate the early completion of tasks and adjust the CPU frequency accordingly.

Contribution of the paper This work is a complete generalization of (Gaujal et al., 2005) where the special case of energy minimization for tasks with FIFO constraints has been studied. Here, a solution for the general case is provided, *i.e.* the FIFO assumption is removed. This generalization requires the introduction of several new concepts, such as the Hasse diagram for inclusion and increasing subsequences in random

sets. The worst case complexity of the proposed algorithm is always better than existing approaches.

Furthermore, we provide a probabilistic analysis of the complexity when the tasks are random. We show that the average complexity of our solution is always lower than $O(N^2\sqrt{N})$. Actually, this bound is not tight because it does not take into account the fact that deadlines have to be larger than arrival times. In the case of Poisson arrivals and exponentially distributed latencies, we prove that the asymptotic complexity for finding the optimal voltage schedule is $O(N^2 \log(N))$. In more general cases, numerical simulations suggest that the $O(N^2 \log(N))$ bound still holds (to be compared with the $O(N \log(N))$ bound in the FIFO case).

Model of the system We consider a single CPU dedicated to the execution of a finite set of some non-recurrent independent tasks (or Íjobsí) with real-time constraints. The CPU processing speed can vary over a continuous range from 0 to 1 (the maximal speed of the processor is fixed to 1 with no loss of generality: this can be achieved by re-scaling the size of the jobs). The set of tasks is $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$ and each task τ_i is characterized by a triplet (a_i, s_i, d_i) where the quantities a_i, s_i, d_i denote the arrival time, the size (time needed to execute the task at maximum speed) and the deadline of task τ_i, respectively. Although only non-recurrent tasks are considered in the following, it is worth noting that the results presented in this paper can be applied to periodic tasks by computing the schedule over one hyper-period, *i.e.* a time interval equal to the Least Common Multiple (LCM) of the periods if tasks are synchronous (first instances of all tasks released simultaneously) or twice the LCM plus the maximum offset between the first instances otherwise (see (Leung and Whitehead, 1982)).

Organization of the paper The Yao et al.'s algorithm for energy minimization is recalled in Section 2. Section 3 is devoted to our solution for finding the critical interval under EDF. The correctness of the algorithm is proved and its complexity is assessed both deterministically and probabilistically. Finally, in Section 4, we address the problem of computing the optimal voltage schedule.

2. Existing results

In this section, we recall the Yao et al.'s algorithm for energy minimization. This construction is presented in details using the notion of *time compression* so that the differences with our approach can be easily highlighted and also because several concepts and notations presented here will be used in subsequent sections. In (Stankovic et al., 1998), an algorithm for testing the feasibility of a set of jobs scheduled under EDF is presented; it will be called the Spuri's algorithm in the following since it is based on a theorem stated by Spuri (Theorem 3.5 pp 33 in (Stankovic et al., 1998)). The Spuri's algorithm is used at step 2 of the Yao et al.'s algorithm as described in paragraph 2.2. The speed of the CPU at time t is denoted $v(t)$ (or $u(t)$, in the following sections, to make a distinction with the classical construction). The construction of the optimal speed function, denoted by $v_{\text{Yao}}(t)$, is given by Yao et al. in (Yao et al., 1995). The goal of this construction is to minimize energy consumption while ensuring that no deadline will be missed. Basically, the algorithm works by identifying the time interval, termed the *critical interval*, over which the highest processing speed is required. The lowest admissible speed is computed for this interval, the tasks belonging to this interval (i.e. arrival date and deadline inside the interval) are then removed (this step is termed *time compression* in the following) and a similar sub-problem is constructed with the remaining tasks.

2.1. Time compression

Here, we explain how to suppress the interval $[a, d]$ from the time-line and how to modify the set of tasks accordingly. The bounds of the interval are denoted by a and d since, in the following, a and d will necessarily be arrival dates and deadline dates respectively. The notation $\tau_i \subseteq [a, d]$ means that task τ_i belongs to $[a, d]$ ($a_i \geq a, d_i \leq d$). When suppressing $[a, d]$, the task set is modified in the following manner :

- If $\tau_i \subseteq [a, d]$ then τ_i is removed.
- For all remaining tasks, s_i is unchanged and

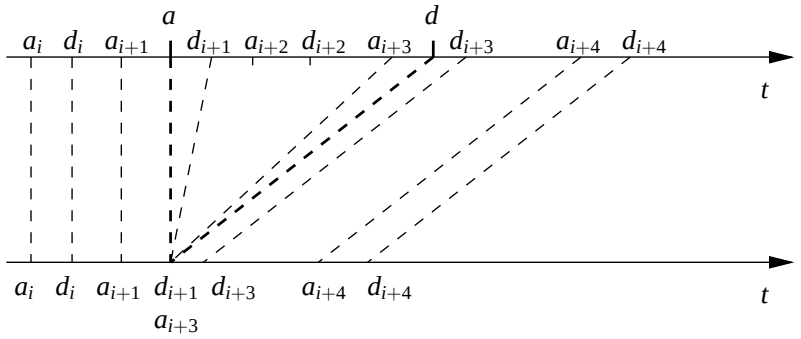


Figure 1. Example of time compression by $[a, d]$. Task τ_{i+2} is removed.

- if $a_i \in [a, d]$ then $a_i := a$, else if $a_i \geq d$ then $a_i := a_i - (d - a)$,
- if $d_i \in [a, d]$ then $d_i := a$, else if $d_i \geq d$ then $d_i := d_i - (d - a)$.

This procedure is called *time-compression by the interval* $[a, d]$ and is illustrated in Figure 1.

2.2. Yao et al.'s algorithm

The Yao et al.'s algorithm is based on a recursive procedure which can also be described by a new sort of ¨compositions" of functions. Let f and g be real functions with appropriate domains and let $[a, d]$ be a time interval, then the function $g \overset{[a,d]}{\vee} f$ is defined as follows :

- if $t < a$, then $(g \overset{[a,d]}{\vee} f)(t) = g(t)$,
- if $t \in [a, d]$, then $(g \overset{[a,d]}{\vee} f)(t) = f(t)$,
- if $t > d$, then $(g \overset{[a,d]}{\vee} f)(t) = g(t - (d - a))$.

Note that this operation is not associative. Also, the domain of all the functions involved is not explicit in the definition. If we define $h := g \overset{[a,d]}{\vee} f$ this means that the domain of f is $[a, d]$, g is defined over an interval including a, say $[b, c]$ and that h is defined over $[b, c + d - a]$.

Yao et al. define the ¨intensity¨ of an interval as the workload brought by the tasks belonging to the interval divided by the length of the interval. Intuitively, it is the smallest amount of work that has to

be done during the interval to meet the timing constraints. The intensity $W_{[a,d]}$ over $[a,d]$ is $W_{[a,d]} = \sum_{\{\tau_k \subseteq [a,d]\}} s_k/(d-a)$. The algorithm proposed in (Yao et al., 1995) constructs the function $v_{\text{Yao}}(t)$ which is the optimal speed for minimizing the total energy consumption while respecting deadlines whatever the scheduling policy. This algorithm can be decomposed in the following steps (starting with $n = 1$) :

1. *Compute the critical interval* $I_n := [a_i, d_j]$ *where* a_i *and* d_j *are such that* $W_{[a_i,d_j]} := \max_{0 \le a \le d} W_{[a,d]}$. *The only values of* a *and* d *that have to be considered are arrival dates and deadlines, respectively.*
2. *Over* I_n, *the function* f_n *is set to a constant :* $f_n(t) := W_{[a_i,d_j]}$, $\forall t \in I_n$. All jobs in I_n must be executed at this rate.
3. *Use time-compression by* I_n. *If at least one task remains then* $n := n+1$ *and go to step 1, otherwise go to step 4.*
4. *Function* v_{Yao} *is completely defined by :* $v_{Yao} := f_n \overset{I_{n-1}}{\vee} (\cdots \overset{I_1}{\vee} f_1)$.

2.3. Some comments on the complexity

The immediate worst-case complexity of this algorithm is $O(N^3)$ where N is the number of jobs : there are at most N successive critical intervals and the computation of each critical interval (step 2) using the Spuri's algorithm can be done in $O(N^2)$ since for each arrival date there are at most N deadlines to consider. In (Yao et al., 1995), the authors write, without more details, that using "a suitable data structure such as the segment tree", the running time can be reduced to $O(N \log^2(N))$. However, this was never really achieved as mentioned in (Yao, 2003). We actually do not know how to obtain an implementation of their algorithm in less than $O(N^3)$ and we are not aware of any paper in literature that has addressed this problem with a complexity lower than $O(N^3)$.

It is worth to point out that feasibility under EDF can actually be assessed using a discrete event simulation of the scheduling using the a_i and the d_i as event points. Such a simulation would run in $O(N \log(N))$ with, for instance, a tree data structure to store the active

jobs. However such a strategy cannot be applied in our context since we need to identify the critical interval.

The rest of the paper is almost entirely devoted to the presentation of a new algorithm that identifies the critical interval faster than existing algorithms (developed independently in (Yao et al., 1995) and (Stankovic et al., 1998)).

3. Computation of the critical interval

The new algorithm presented here relies on the construction of the Hasse diagram of a Partially Ordered SET (Poset) and on the solution of a shortest path problem that has been proposed in (Gaujal et al., 2005). These two points will be investigated first. We then provide a new algorithm that constructs a CPU speed function $u^*(t)$ and we prove that $u^*(t) = v_{\text{Yao}}(t)$ over the critical interval. Then, we evaluate the algorithmic complexity of the algorithm both in a deterministic and in a probabilistic way and show that it improves upon Yao et al's algorithm.

3.1. The Hasse diagram

Let us consider the Poset P on the set of tasks with the partial *strict order* being defined by the strict inclusion of the intervals : $P = \{[a_i, d_i], \prec\}$, where the strict order $\prec$ corresponds to the level of nesting of the intervals. More precisely, $[a, d] \prec [a', d']$ if $[a', d'] \subset [a, d]$, *i.e.* $a' > a$ and $d' < d$.

A finite Poset can be represented by its Hasse diagram (see (Weisstein, 1999) for more details), where the elements of the Poset are partitioned into *levels*. The first level is the subset of all minimal elements (here intervals which are not included in any other), the second level is the subset of all elements which are immediately larger than the minimal elements (i.e. included in at least one minimal interval) and so forth. The levels in the diagram are denoted $L_1, \cdots, L_K$ where K is the highest level. One denotes by $L(i)$ the level of the interval $[a_i, d_i]$ in P, it will be called the level of task τ_i. The set of tasks that will be considered for illustration purposes in the following is :

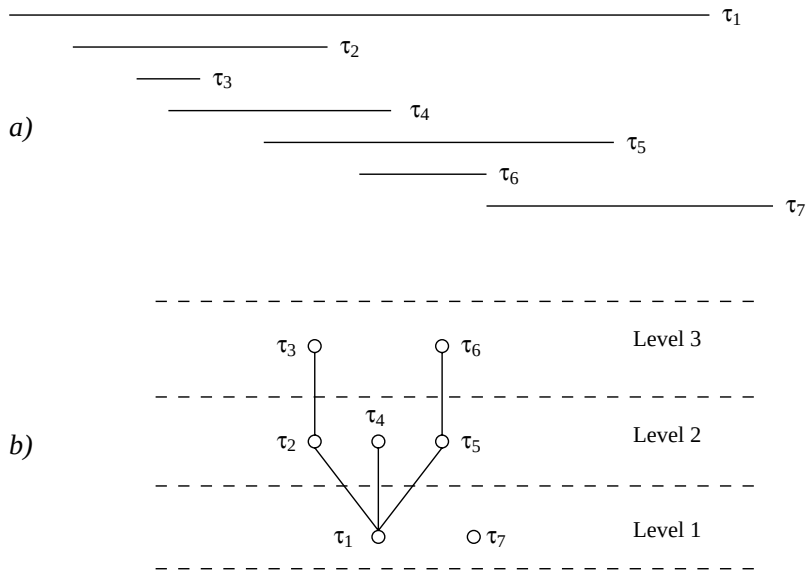


Figure 2. Sub-figure a) represents the overlapping of the intervals $[a_i, d_i]$ while sub-figure b) shows the corresponding Hasse diagram.

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
a	0	2	4	5	8	11	15
d	22	10	6	13	19	15	24
s	3	1	1	4	1	2	1

The Hasse diagram associated with this set of tasks is given in Figure 2 b) while Figure 2 a) shows how the intervals $[a_i, d_i]$ overlap.

We now explain how to compute the levels of all the tasks in the Hasse diagram in $O(N \log(N))$ elementary operations. The construction uses an array L that will be used to store the levels of the tasks : $L[i]$ is the level of τ_i. Another array, called T, stores the value of largest deadline on a given level: $T[k] = \max\{d_i s.t.$ level of $\tau_i = k\}$. The arrays L and T are constructed iteratively by the algorithm below.

1. *Reorder the set of tasks so that they are sorted according to the arrival times a_i (ties are broken using the earliest deadlines).*

2. $T[0] := \infty$, $T[1] := d_1$, $L[1] := 1$ *and for all* $k = 2..N$ *set* $T[k] := 0$.

3. *For i from 2 to N do*

   a) *Dichotomy-Search of d_i in the current T gives k, the only index such that $T[k] > d_i \geq T[k+1]$.*

   b) $T[k+1] := d_i$; $L[i] := k+1$.

Note that the dichotomic search is valid because T is sorted in increasing order. The correctness of this construction comes from the fact that, once the arrivals have being sorted ($a < a'$), checking if $[a, d] \subsetneq [a', d']$ is done by merely checking if $d' > d$ (see, for example, (Fishburn, 1985) for more on interval orders).

The total complexity of the algorithm is $O(N \log(N))$: the initial sort is in $O(N \log(N))$ and each dichotomy uses $O(\log(N))$ operations and it is repeated N times.

3.2. The shortest path construction

The case $K = 1$ (FIFO tasks) is treated in (Gaujal et al., 2005). The algorithm involves the construction of the shortest path between 0 and T with given upper and lower bounds. The construction in the general case also uses this framework, but, this time, we need to construct such a path for each level of the Hasse diagram.

For convenience, we assume that $a_1 = 0$ (all dates can be shifted to the left) and let $\mathbf{T} \stackrel{\text{def}}{=} \max_i\{d_i\}$. For all tasks on level k and higher in the Hasse diagram, we construct two cumulative functions $A_k(t)$ and $D_k(t)$ over the interval $[0, \mathbf{T}]$ using the following definition:

$$A_k(t) = \sum_{i,\ L(i)\geq k} s_i \cdot \mathbf{1}_{[a_i<t]}, \quad D_k(t) = \sum_{i,\ L(i)\geq k} s_i \cdot \mathbf{1}_{[d_i\leq t]}.$$

The functions $A_k(t)$ and $D_k(t)$ are staircase functions (*i.e.* piece-wise constant, with a finite number of pieces). Also note that A is left-continuous while D is right-continuous. They represent the cumulative work arrival and the cumulative work deadline, respectively.

Let g be an arbitrary non-decreasing and strictly convex function. In power aware scheduling, such a function is classically used to model the energy consumption of the processor. The consumption at time t

depends on the processor speed $u(t)$ and, for instance, with the CMOS technology, typically, $g(u(t)) \approx \alpha C u(t)^{1+2/(\gamma-1)}$, where $1 \leq \gamma \leq 3, \alpha \geq 0, C \geq 0$, see (Gruian, 2002) for more details.

Let us consider Problem 1 where the parameter k corresponds to the subset of tasks with levels k and higher in the Hasse diagram.

Problem 1. *Find an integrable function* $u_k^* : [0, \boldsymbol{T}] \rightarrow \mathbb{R}$ *such that* $\int_0^{\boldsymbol{T}} g(u_k^*(s))ds$ *is minimized, under the constraints*

$$u_k^*(t) \geq 0 \quad \forall t \in [0, \boldsymbol{T}], \tag{1}$$

$$\int_0^t u_k^*(s)ds \leq A_k(t) \quad \forall t \in [0, \boldsymbol{T}], \tag{2}$$

$$\int_0^t u_k^*(s)ds \geq D_k(t) \quad \forall t \in [0, \boldsymbol{T}]. \tag{3}$$

Constraint (1) means that the processor speed is necessarily non-negative. Constraint (2) simply states that one cannot execute the work that has not arrived while constraint (3) says that the total amount of work done by time t shall not be smaller than the cumulated work of the tasks with deadlines lower than or equal to t.

It has been shown in (Gaujal et al., 2005) that if there exists a function $u_k(t)$ that verifies (1), (2) and (3) then an EDF schedule will meet the deadline constraints if all tasks are FIFO ($a_i \leq a_j \rightarrow d_i \leq d_j$). This does not hold for arbitrary tasks since the feasibility requirement cannot be solely expressed with constraints (2) and (3). To illustrate this fact, we have constructed two sets of tasks and the corresponding cumulative functions A and D (the index k is skipped). The first set is made of job $\tau_1 = (0, 4, 10)$ and job $\tau_2 = (4, 1, 5)$. The second set is $\tau_1' = (0, 1, 5), \tau_2' = (0, 3, 10)$ and $\tau_3' = (4, 1, 10)$. Note that the second set is FIFO while the first one is not. The cumulative work arrivals $A(t)$ and $A'(t)$ are the same in both cases. The cumulative deadlines $D(t)$ and $D'(t)$ are also equal. Therefore, the two solutions of Problem 1 with the two sets of tasks, $u^*(t)$ and $u^{*\prime}(t)$ are equal. Figure 3 displays the function $A(t) = A'(t)$ as well as $D(t) = D'(t)$ and the integral $U^*(t) = \int_0^t u^*(s)ds$ where $u^*(t) = 1/2, \quad \forall 0 \leq t \leq 10$.

Note that the CPU speed $u^*(t)$ is sufficient to finish all tasks of the second set before their deadlines but is not enough for the first set :

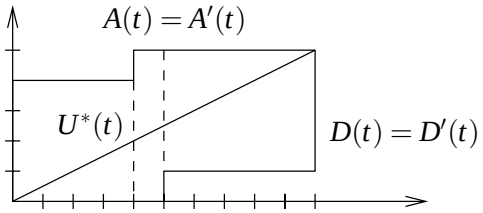


Figure 3. FIFO and non-FIFO tasks

task τ_2' needs a CPU speed greater than one during the interval of time $[4, 5]$ to meet its deadline.

Problem 1 has been studied in (Gaujal et al., 2005). Solving problem 1 with $k = 1$ provides the optimal voltage schedule u^* that ensures feasibility for FIFO tasks. The main properties of the solution of Problem 1 are summarized in the following theorem.

Theorem 1. *(*Gaujal et al., 2005*)*
i- For all k, the optimal solution u_k^* of problem 1 is unique if g is strictly convex (up to a set of measure 0). Furthermore, u_k^* does not depend on g, as long as g is a non-decreasing convex function.
ii- The optimal solution u_k^* satisfies the following inequality[1]:
$\sup_{0\leq t\leq \boldsymbol{T}} u_k^*(t) \leq \sup_{0\leq t\leq \boldsymbol{T}} u_k(t)$, *for all functions u_k satisfying constraints* (1), (2) *and* (3).
iii- The integral $U_k^* \stackrel{\text{def}}{=} \int_0^t u_k^*(s)ds$ is the shortest path from 0 to $\boldsymbol{T}$ while staying between A_k and D_k (see Figure 4 for an illustration).
iv- If the functions A_k and D_k are given under the form of two ordered lists of points, the construction of $u_k^*(t)$ can be done in linear time.

The algorithm constructing u_k^*, presented in (Gaujal et al., 2005), is inspired from the "Graham scan" algorithm which is an algorithm for computing convex hulls that runs in linear time when applied to a set of ordered points in the plane (see, for instance, (Boissonnat and Yvinec, 1998)).

[1] here, the sup operator stands for the essential supremum, since all functions are only defined almost everywhere.

3.3. Description of the critical interval algorithm

Let us recall that K is the highest level in the Hasse diagram, one denotes by $\mathcal{T}_k$, with $1 \leq k \leq K$, the set of all the tasks at level k and higher. The algorithm for finding the critical interval is given below.

1. *For all* $k \in \{1 \cdots, K\}$
   a) *Construct* $u_k^*(t)$ *the optimal solution of Problem 1 with parameter* k.
   b) *Define* $r_k := \sup_{0 \leq t \leq \boldsymbol{T}} u_k^*(t)$ *and* I_k *an interval s.t.* $\forall t \in I_k,\ u_k^*(t) = r_k$.
2. *Let* $k_c := \ argmax_{1 \leq k \leq K} r_k$. *Then* I_{k_c} *is a critical interval.*

An example of this construction is given in Figure 4 using the same set of tasks as in Figure 2. The functions $U_i^*(t) \stackrel{\text{def}}{=} \int_0^t u_i^*(x)dx$ are displayed instead of $u_i^*(t)$ in order to show why such functions are called shortest paths. We get $r_3 = 1/2$, $r_2 = 7/11$ and $r_1 = 12/22$. Note that the maximum $r_2 = 7/11$, is reached by u_2^* over the interval $[4, 15]$ made of tasks τ_3, τ_4 and τ_6. This is the critical interval. Also note that the whole set of tasks is feasible since $r_2 = 7/11 < 1$.

3.4. Correctness of the algorithm

This section is devoted to the proof of the correctness of the algorithm. In the following, I_c denotes the critical interval and W_{I_c}, the intensity over the critical interval, is denoted W_c. We also define the function v_{Yao}^k, as the Yao's construction over the set of tasks $\mathcal{T}_k$ (therefore, $v_{\text{Yao}}^1 = v_{\text{Yao}}$).

Lemma 2. *The following assertions are true:*
i- $\sup_t v_{Yao}(t) = W_c$.
ii- For all k, v_{Yao}^k *is a solution of Problem 1 with parameter* k *(not necessarily optimal).*
iii- For all $1 < k \leq K$, $\sup_t v_{Yao}^k(t) \leq \sup_t v_{Yao}^{k-1}(t)$.

Proof. The proof of point i is a direct consequence of the construction of v_{Yao} (see (Yao et al., 1995)). Point ii follows from the fact that all

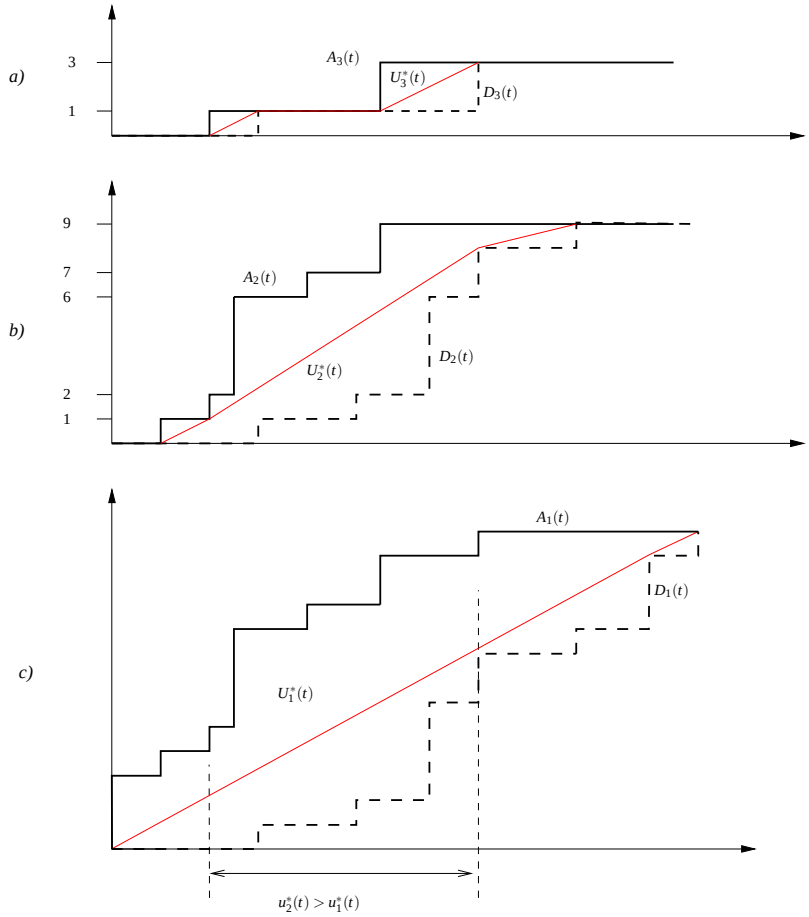


Figure 4. Figure a) shows $U_3^*(t) = \int_0^t u_3^*(x)dx$, Figure b) shows $U_2^*(t) = \int_0^t u_2^*(x)dx$ and Figure c) shows $U_1^*(t) = \int_0^t u_1^*(x)dx$.

feasible scheduling policies with a processor with speed u must satisfy constraints 1,2 and 3. As for *iii*, this is simply because $\mathcal{T}_k \subseteq \mathcal{T}_{k-1}$ and because the maximal intensity grows when new tasks are added. □

Lemma 3. *For all* $t \in [0,T]$, $\quad \sup_t u_k^*(t) \leq \sup_t v_{Yao}^k(t)$.

Proof. By Lemma 2(*ii*), v_{Yao}^k is a solution of Problem 1 with parameter k (not necessarily optimal). Using Theorem 1(*iii*) shows that $t \in [0,T]$, $\quad \sup_t u_k^*(t) \leq \sup_t v_{\text{Yao}}^k(t)$. □

Lemma 4. $\sup_k r_k \geq W_c$.

Proof. We consider all the tasks included in the critical interval $I_c = [a_c, d_c]$ as one super-task τ_c whose size s_c is the sum of the sizes of all tasks included in I_c while its arrival is a_c and its deadline is d_c. Note that the level L_c of this new task is the lowest level of all tasks in I_c.

By definition of τ_c, all non-critical tasks in $\mathcal{T}_{L_c}$ are not included in τ_c : non-critical tasks starting before τ_c end before τ_c and non critical tasks starting after a_c finish after d_c. This means that $D_{L_c}(d_c) - A_{L_c}(a_c) = 0$ since A is left-continuous and D right-continuous. The construction of $u^*_{L_c}$ implies that $\sup_{t \in I_c} u^*_{L_c}(t) \geq (D_{L_c}(d_c) - A_{L_c}(a_c))/(d_c - a_c) = W_c$. Finally, $\sup_k r_k \geq r_{L_c} \geq W_c$. □

Theorem 5. *Let $k_c = \ argmax_k\ r_k$. Then, $r_{k_c} = W_c$ and the interval I_{k_c} is a critical interval.*

Proof. By Lemma 2(i), $\sup_t v_{\text{Yao}}(t) = W_c$. Moreover, Lemma 2$(ii)$ says that $\sup_t\ v^k_{\text{Yao}}(t) \leq \sup_t\ v^{k-1}_{\text{Yao}}(t)$. Combining this with Lemma 3, one gets $\sup_t\ v_{\text{Yao}}(t) \geq \sup_t\ u^*_k(t)$ for all k. In particular, $\sup_t v_{\text{Yao}}(t) \geq \sup_t u^*_{k_c}(t)$, where $k_c = \ \text{argmax}_k\ r_k$. Finally, Lemma 4 says that $r_{k_c} \geq W_c$ and using the two previous inequalities shows $r_{k_c} = W_c$. The interval I_{k_c} is therefore a critical interval. □

Note that $u^*_k \neq v^k_{\text{Yao}}$ in general. However, as shown above, the equality holds inside the critical interval for k_c.

3.5. Deterministic analysis of the complexity

Assume that tasks have already been sorted by their arrival date as well as by their deadlines. Assume that the levels in the Hasse diagram are also given. The time complexity of constructing u^*_k is linear in the number of tasks at level k and higher as shown in (Gaujal et al., 2005). Computing r_k is also linear in the number of tasks at levels greater than k. Therefore, the complexity of the algorithm to construct v_1 is (up to a multiplicative constant)

$$\sum_{i=1}^{N} L(\tau_i) = \sum_{k=1}^{K} N_i \cdot k$$

where N_i is the number of tasks at level i. The highest complexity corresponds to the case where there is one task at each level lower than K and $N_K = N - (K-1)$ tasks at level K. The complexity is thus $O(KN - K^2/2)$.

Note that this time complexity is linear if all tasks are on level 1 (the FIFO case where $K = 1$) which is the best possible case. The worst case arises when $K = N$ with 1 task on each level, where the complexity becomes $O(N^2/2)$ (it is easy to patch this worst case and solve the problem in linear time by merely testing $r_1 \leq 1$). Since $K \leq N$, the complexity of the algorithm presented above is at most quadratic. We would like to point out that a worst case complexity of $O(NK)$ constitutes an improvement over $O(N^2)$ (complexity of the Spuri's algorithm) since K is in most cases much smaller than N (it depends on structural properties of the task set and not directly on its size). The problem investigated below is to estimate the value of K for typical sets of tasks.

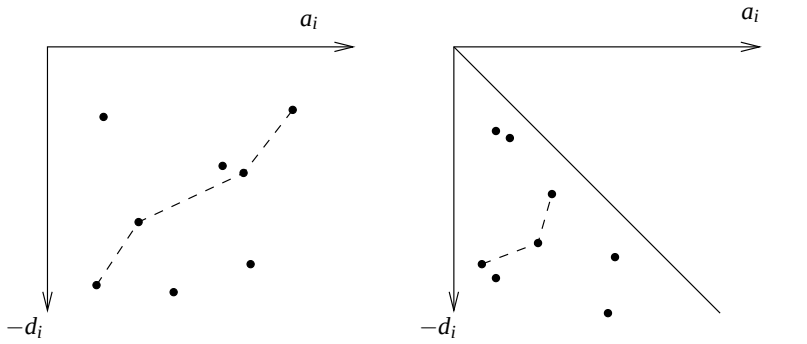
3.6. Probabilistic analysis of the complexity

As seen above, the main improvement provided by the new algorithm is to replace a factor N by a factor K in the complexity. Although, it is obvious that K is always smaller than or equal to N, it is interesting for practical purposes to know what the difference may be in typical cases.

3.6.1. *Bound on the average complexity for arbitrary distributions*

In this paragraph, we provide upper bounds on K when the characteristics of the tasks are chosen randomly. Recall that K is the number of levels in the Hasse diagram of the N tasks. In other words, K is the length of the longest increasing subsequence in the Poset P (see paragraph 3.1). Here, the Poset P can be represented in $\mathbb{R}^2$ by comparing the points $(a_i, -d_i)$ using the component wise ordering. An illustration of this representation is given in Figure 5-b).

The problem to estimate the average length of the longest increasing subsequence in a two-dimensional partially ordered set was studied extensively in (Aldous and Diaconis, 1995). If the arrivals and the dead-



a) Points are randomly chosen in the whole quadrant. The longest increasing sequence is of length 4.

b) points are randomly chosen below the diagonal. The longest increasing sequence is shorter (3).

Figure 5. The general case versus the case of jobs where necessarily $d_i \geq a_i$.

lines (a_i, d_i) are chosen randomly in $\mathbb{R}^2_+$, and if all tasks are mutually independent, then one has the following result.

Theorem 6. *(*Aldous and Diaconis, 1995*) The expected value of K verifies $\mathbb{E}(K) = 2\sqrt{N} + o(\sqrt{N})$.*

This result is quite general since no assumptions are made on the distributions of the points in $\mathbb{R}^2_+$. In our case, we can hope that the expected value of K is much smaller because the deadlines and the arrival times verify the constraint $(\forall i, \quad d_i \geq a_i)$ so that all points $(a_i, -d_i)$ are below the diagonal (see Figure 5b).

When jobs are considered instead of random points, the average K should be much smaller than when the points can be anywhere in $\mathbb{R}^2$ as in Figure 5. To make this intuition more precise, we consider the case where the arrival process is Poisson and the relative deadlines $(d_i - a_i)$ have exponential distributions.

3.6.2. *Average complexity for Poisson arrivals and exponential deadlines*

Lemma 7. *If the arrival process is Poisson and the relative deadlines have exponential distributions, then $\mathbb{E}(K) = O(\log(N))$.*

Proof. The intensity of the Poisson process is denoted by μ and the parameter of the exponential deadlines by λ. As mentioned before, the problem can be seen as a two dimensional problem by plugging a_i on the x-axis and $-d_i$ on the y axis. We obtain a set E of N points in the plane. In this framework, K is the longest sequence of component-wise increasing points $(a_{i_1}, -d_{i_1}) < \cdots < (a_{i_K}, -d_{i_K})$. Now, if we consider any given point $h = (a_h, -d_h)$, the longest increasing sequence S_h starting in this point is clearly smaller that the total number T_h of points which are larger than $(a_h, -d_h)$. Let H be a typical random point.

$$\begin{aligned}
\mathbb{P}(S_H \geq b) &\leq \mathbb{P}(T_H \geq b), && (4)\\
&\leq \int_{x=0}^{\infty} \lambda e^{-\lambda x} \sum_{k=b}^{\infty} \mathbb{P}(k \text{ arrivals in an interval of length } x)\, dx, && (5)\\
&= \lambda \sum_{k=b}^{\infty} \int_{x=0}^{\infty} e^{-(\lambda+\mu)x} \mu^k x^k / k!\, dx, && (6)\\
&= \frac{\lambda}{\lambda+\mu} \sum_{k=b}^{\infty} \left(\frac{\mu}{\lambda+\mu}\right)^k \int_{y=0}^{\infty} e^{-y} y^k / k!\, dy, && (7)\\
&= \frac{\lambda}{\mu} \sum_{k=b}^{\infty} \left(\frac{\mu}{\lambda+\mu}\right)^{k+1} = \left(\frac{\mu}{\lambda+\mu}\right)^b, && (8)
\end{aligned}$$

where (5) comes from conditioning on the value of $d_H = x$ and the fact that all points larger than H must have an arrival time between a_H and $a_H + x$. Equality between (5) and (6) holds by applying the monotone convergence theorem and $y = (\lambda + \mu)x$ in (7).

Next, we use the fact that S_H are associated variables (see Appendix A for a proof). By using Corollary 3.3 in (Barlow and Proschan, 1981),

$$P(K > b) = P(\max_h S_H > b) \leq 1 - \left(\prod_{h \in E} (1 - P(S_H > b)) \right).$$

By using (8),

$$1 - \left(\prod_{h \in E} (1 - P(S_H > b)) \right) \leq 1 - (1 - (\left(\frac{\mu}{\lambda+\mu}\right)^b)^N.$$

Now, introducing $\alpha = \log \frac{\lambda+\mu}{\mu}$, we obtain

$$\begin{aligned}
\mathbb{E}(K) &= 1 + \int_0^\infty P(K > x)dx \\
&\leq 1 + \int_0^\infty 1 - (1 - e^{-\alpha x})^N dx \\
&= 1 + \int_0^\infty (1 - \sum_{k=0}^{N} (-1)^k C_n^k e^{-\alpha k x})dx = 1 + \int_0^\infty \sum_{k=1}^{N} (-1)^{k+1} C_n^k e^{-\alpha k x} dx \\
&= 1 + \sum_{k=1}^{N} (-1)^{k+1} C_n^k \int_0^\infty e^{-\alpha k x} dx = 1 + \sum_{k=1}^{N} (-1)^{k+1} C_n^k \frac{1}{\alpha k} \\
&= 1 - \frac{1}{\alpha} \sum_{k=1}^{N} (-1)^k C_n^k \int_0^1 u^{k-1} du = 1 - \frac{1}{\alpha} \int_0^1 \frac{1}{x} (\sum_{k=0}^{N} (-1)^k C_n^k u^k - 1) du \\
&= 1 - \frac{1}{\alpha} \int_0^1 \frac{1}{u} ((1-u)^N - 1) du = 1 - \frac{1}{\alpha} \int_0^1 \frac{1}{u} (1 - u - 1) \sum_{i=0}^{N-1} (1-u)^i du \\
&= 1 + \frac{1}{\alpha} \int_0^1 \sum_{i=0}^{N-1} (1-u)^i du = 1 + \frac{1}{\alpha} \sum_{i=0}^{N-1} [\frac{1}{i+1} (1-u)^{i+1}]_{u=0}^1 \\
&= 1 + \frac{1}{\alpha} \sum_{i=1}^{N} \frac{1}{i} = O(\log(N)).
\end{aligned}$$

□

Although the result is only proved in the Poisson-exponential case (i.e. task arrivals are Poisson distributed and relative deadlines are exponentially distributed), we believe that the $\log(N)$ bound for the average K is rather robust and holds in many more cases. To illustrate this, we have run several simulations where the set of jobs were randomly chosen according to various probability distributions.

3.6.3. *Experimental results*

To test the robustness of the $O(\log(N)$ bounds over the distribution of the arrivals and the deadlines, the average value of K was computed on sets of jobs whose characteristics (arrivals and deadlines) are randomly chosen. Several probability laws were considered such that the intensity of the arrivals and the expected value for relative deadlines are identical for all of them. The following configurations were considered :

- uniformly distributed inter-arrivals / uniformly distributed relative deadlines,
- uniformly distributed inter-arrivals / exponentially distributed relative deadlines,
- Poisson arrivals / exponentially distributed relative deadlines,
- Poisson arrivals / power-law distributed relative deadlines (*i.e.* $P[X > x] = Cx^{-n}$. Here, $n = 2$ and C is adjusted to obtain the same mean as in the other cases).

Figure 6 summarizes the results. The results of the case "uniformly distributed arrivals / exponentially distributed relative deadlines", which are almost identical to "Poisson arrivals / exponentially distributed relative deadlines", are not shown in Figure 6. Each point is the average value of 1000 experiments. One observes that, in these experiments, the $\log(N)$ bound for K holds for all the distributions. Also observe that the average value of K increases with the variance of the deadlines (from uniform, to Poisson, to power laws). However, in all cases the behavior of K remains logarithmic in N (even when the variance is infinite, as for the power law). This suggests that the $\log(N)$ bound should hold for many more cases than those covered by Lemma 7.

Now, if one considers the average complexity of the complete algorithm for computing the critical interval, Theorem 6 and Lemma 7 imply that the average complexity of the new algorithm is lower than or equal to $O(N\sqrt{N})$ and can be $O(N \log N)$ under the assumptions used in Lemma 7. On the other hand, the *average* complexity of the classical algorithm of Spuri is larger than $k((N^2 + N)/2)$ for some constant k, whatever the distribution of the arrivals and the deadlines, since this is the minimal number of operations required to find the critical interval (corresponding to the case where tasks have FIFO constraints).

3.7. Parallel complexity

Yet another advantage of the shortest path algorithm over Spuri's algorithm, is the possibility to run it in parallel efficiently. Here is a parallel version of the algorithm:

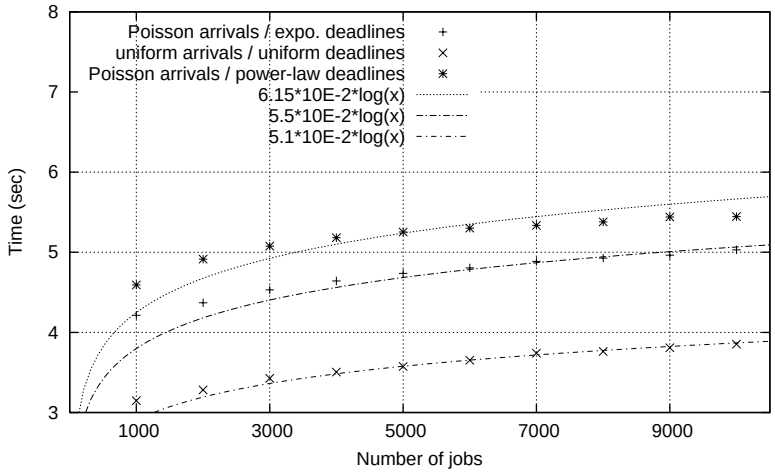


Figure 6. Average value of K, the number of levels in the Hasse diagram, for different distributions of the arrivals and relative deadlines (same expected values for all distributions).

1. *In parallel for all* $k \in \{1 \cdots, K\}$

   a) *Construct* $u_k^*(t)$ *the optimal solution of Problem 1 with parameter* k.

   b) *Define* $r_k := \sup_{0 \le t \le \boldsymbol{T}} u_k^*(t)$ *and* I_k *an interval s.t.* $\forall t \in I_k,\ u_k^*(t) = r_k$.

2. *Compute* $k_c := argmax_{1 \le k \le K}\ r_k$ *by comparing the values of* r_k, $k \in \{1 \cdots, K\}$ *with a binary tree.*

Using the PRAM model, with $P \le K$ processors, the complexity of the first step is $O(NK/P)$ while the complexity of the second step is $O(K/P + \log(P))$. This yields an almost linear speed up.

To go further, one may also compute each $u_k^*(t)$ in parallel within step (1), using a parallel algorithm for computing shortest paths in graphs in time $O(log^2(n))$ with a polynomial number of processors. This means that the whole algorithm belongs to the class NC of parallel algorithms.

On the other hand, we do not know any efficient way to parallelize Spuri's algorithm and we are not aware of any paper in the literature that has addressed this problem.

4. Energy minimization

In this section, we consider the problem of choosing, at each time t, the speed $u(t)$ to execute all tasks within their deadline constraints while minimizing the total energy consumption.

4.1. Description of the algorithm

The function v_{Yao} actually provides a solution to the problem of minimizing the total energy used by the processor when the immediate energy consumption of a processor going at speed u, is of the form $g(u)$, where g is convex and increasing. This result is obtained by Yao et al. in (Yao et al., 1995). Let us now describe how to compute the sequence of optimal frequencies with a lower complexity than the Yao et al.'s algorithm.

The scheme is exactly the same as in (Yao et al., 1995) but we substitute the Spuri's algorithm for the one proposed in Section 3 that finds the critical interval. Then the speed is fixed over the critical interval. Time compression by the critical interval is used and a sub-problem is constructed with the remaining tasks. This leads to the following algorithm.

1. *For* $k = 1..K$ *construct* $u_k^*(t)$. *Let* $r_i = \sup_{0 \le t \le T} u_k^*(t)$; *the critical interval* $I_c = [a_i, b_j]$ *is such that* $W_{[a_i,b_j]} = \sup_k r_k$.

2. *Set the speed on* $[a_i, b_j]$ *equal to* $W_{[a_i,b_j]}$. *Using time compression, remove interval* $[a_i, b_j]$. *Return to step 1 if at least one task remains.*

The execution of the algorithm in our foregoing example, whose task set is given in the table of paragraph 3.1, is displayed in Figure 7.

It should be noted that K, the number of levels in the Hasse diagram, cannot increase after the time compression of the critical interval. Indeed, if task, say τ_1, is not strictly included in task, say τ_2, before time compression, then τ_1 is not strictly included in task τ_2, after compression, either. On the other hand, K may decrease by time compression. Imagine that task τ_1 is strictly included in task τ_2 before time compression and that the deadlines (or the arrival times) of both tasks fall in

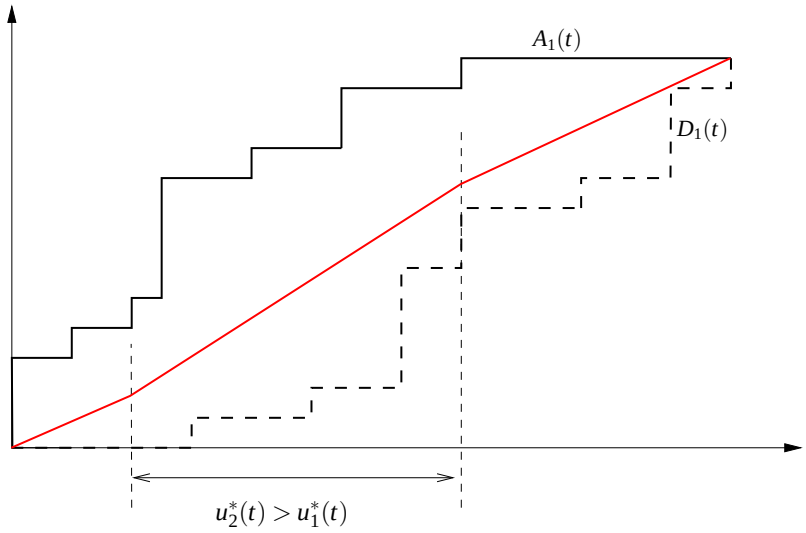


Figure 7. Minimal energy consumption constructed using the repeated shortest path algorithm.

the compressed interval, then they are not strictly included anymore after time compression.

Since there is at most N critical intervals, the feasibility algorithm (step 1) is invoked at most N times and the worst-case complexity of the algorithm is $O(KN^2)$. As for the average complexity, by Theorem 6, it is always lower than or equal to $O(N^2\sqrt{N})$. Furthermore, when tasks have Poisson distributed arrival times and exponential latencies, the average complexity is $O(N^2 \log N)$ by Lemma 7.

The correctness of this algorithm is implied by the correctness of the feasibility algorithm (see Theorem 5) and the correctness of the Yao et al.'s algorithm.

4.2. Experimental results

To evaluate the gain that one can expect in practice with our algorithm (the Shortest Path algorithm) with respect to Yao et al.'s algorithm, experiments were conducted on random sets of jobs. The sets of jobs are created according to the following procedure :

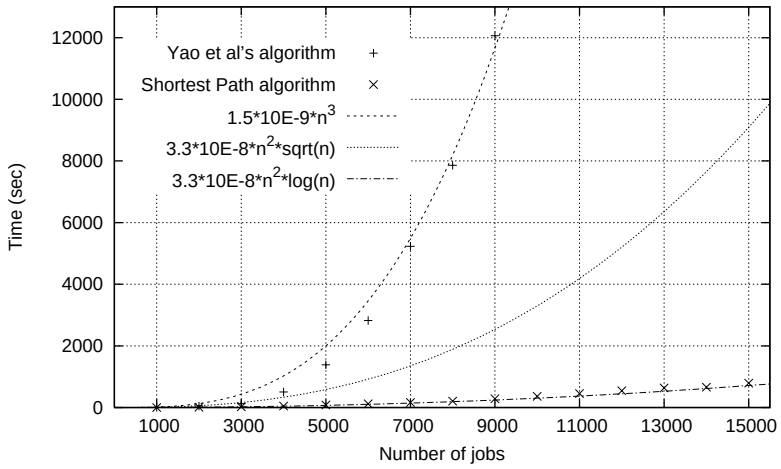


Figure 8. Computation times of the optimal voltage schedule for the Shortest Path and Yao et al. algorithms on the same random sets of tasks. For each problem size, the result is the average of 10 experiments. Computations were stopped when at least one experiment took more that three hour of CPU time.

- the arrivals follow a Poisson process of mean S/N where S is an arbitrary time interval ($S = 2^{30}$ in our experiments) and N is the number of jobs,

- the deadline of each job follows an exponential distribution of mean S/N while the size is uniformly distributed over an interval in such a way as to reach an average load of 0.5.

The computation times for each algorithm are shown in Figure 8 for a number of jobs varying between 1000 and 15000. Each point is the average value of ten experiments. Computations were stopped when at least one experiment took more than three hour of CPU time on a 2Ghz CPU and no point is drawn in this case (*i.e.* Yao et al.'s algorithm with more than 9000 jobs).

From Figure 8, one sees, as expected, that the running time of Yao et al.'s algorithm is $O(N^3)$ while our shortest path algorithm runs in $O(N^2 \log(N))$ much less than the $O(N^2\sqrt{N})$ bound valid for arbitrary distributions of the tasks' characteristics. The $O(N^2 \log(N))$ behavior was expected since the experiments were conducted with the assumptions of Lemma 7.

When the number of jobs becomes greater than 9000, some solutions cannot be found within three hour of CPU time with Yao et al.'s algorithm while problems with more than hundreds of thousands jobs can be solved in the same amount of time using the Shortest Path algorithm. For 9000 jobs, the speedup is equal to 42 with respect to Yao et al.'s algorithm.

While several thousands may seem like a rather large number of tasks, it has to be noted that in practice, it is sometimes necessary to handle such large sets of jobs since computing the voltage schedule for periodic tasks with arbitrary deadlines has to be done over a period of time that can be very long (twice the LCM of the tasks periods plus the maximum offset between the first instances of the tasks, see (Leung and Whitehead, 1982)).

5. Conclusions

In this study, we first proposed a new algorithm for finding the critical interval of a set of independent jobs scheduled under EDF. This algorithm outperforms the existing approaches in terms of average and worst-case complexity. In future work, it may be possible that further complexity improvements can be achieved with local optimization procedures when tasks exhibit specific characteristics (for instance, when tasks are almost all nested).

Based on this result, the other contribution of this study is a new way of finding the optimal voltage schedule. This proposal has a lower complexity than the classical algorithm of Yao et al. and it also brings several extensions such as coping with a finite number of speeds, non-convex cost functions or finding the schedule that also minimizes the number of speed changes (see (Gaujal et al., 2005)).

Acknowledgments The authors would like to thank James Martin who pointed out reference (Aldous and Diaconis, 1995) and Jean-Marc Vincent who provided reference (Barlow and Proschan, 1981) useful in the computation of the average complexity.

References

AbouGhazaleh, N., D. Mosse, B. Childers, R. Melhem, and M. Craven: 2003, 'Collaborative Operating System and Compiler Power Management for Real-Time Applications'. In: *9th IEEE Real-Time and Embedded Technology and Applications Symposium*. pp. 133–143.

Aldous, D. and P. Diaconis: 1995, 'Hammersley's interacting particle process and longest increasing subsequences'. *Probability Theory and Related Fields* **103**(2), 199–213.

Aydin, H., R. Melhem, D. Mossé, and P. Mejia-Alvarez: 2004, 'Power-Aware Scheduling for Periodic Real-Time Tasks'. *IEEE Transactions on Computers* **53**(5), 584–600.

Aydin, H., M. R., D. Mossè, and M.-A. P.: 2001, 'Dynamic and Agressive Scheduling Techniques for Power Aware Real-Time Systems'. In: *22th Real-Time Systems Symposium*. pp. 95–105.

Barlow, R. E. and F. Proschan: 1981, *Statistical Theory of Reliability and Life Testing: Probability Models*. To Begin With, second edition.

Boissonnat, J. and M. Yvinec: 1998, *Algorithmic Geometry*. Cambridge University Press.

Fishburn, P.: 1985, *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*, Interscience Series in Discrete Mathematics. Wiley.

Gaujal, B., N. Navet, and C. Walsh: 2003, 'Real-Time Scheduling for Optimal Energy Use'. In: *Journées d'étude Faible Tension - Faible Consommation (FTFC'03)*. pp. 159–166.

Gaujal, B., N. Navet, and C. Walsh: 2005, 'Shortest-path algorithms for real-time scheduling of FIFO tasks with minimal energy use'. *ACM Transactions on Embedded Computing Systems* **4**(4), 907–933. Preliminary version available as INRIA Research Report RR-4886, `http://www.inria.fr/rrrt/rr-4886.html`.

Gruian, F.: 2001, 'On energy reduction in hard real-time systems containing tasks with stochastic execution times'. In: *IEEE Workshop on Power Management for Real-Time and Embedded Systems*. pp. 11–16.

Gruian, F.: 2002, 'Energy-Centric Scheduling for Real-Time Systems'. Ph.D. thesis, Lund Institute of Technology.

Leung, J.-T. and J. Whitehead: 1982, 'On the complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks'. *Performance Evaluation* **2**, 237–250.

Liu, Y. and A. Mok: 2003, 'An Integrated Approach for Applying Dynamic Voltage Scaling to Hard Real-Time Systems'. In: *9th IEEE Real-Time and Embedded Technology and Applications Symposium*. pp. 116–123.

Lorch, J. and A. Smith: 2001, 'Improving dynamic voltage scaling algorithms with PACE'. In: *ACM SIGMETRICS 2001 Conference*. pp. 50–61.

Mossè, D., H. Aydin, B. Childers, and R. Melhem: 2000, 'Compiler-assisted dynamic power-aware scheduling for real-time applications'. In: *Workshop on Compiler and Operating Systems for Low-Power*.

Pillai, P. and K. Shin: 2001, 'Real-Time Dynamic Voltage Scaling for Low-Power Embedded Systems '. *Operating Systems Review* **35**(5), 89–102.

Qadi, A., S. Goddard, and S. Farritor: 2003, 'A Dynamic Voltage Scaling Algorithm for Sporadic Tasks'. In: *24th IEEE International Real-Time Systems Symposium*. pp. 52–62.

Quan, G. and X. Hu: 2001, 'Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors'. In: *Design Automation Conference*. pp. 828–833.

Quan, G. and X. Hu: 2002, 'Minimum Energy Fixed-Priority Scheduling for Variable Voltage Processors'. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*.

Scordino, C. and G. Lipari: 2006, 'A Resource Reservation Algorithm for Power-Aware Scheduling of Periodic and Aperiodic Real-Time Tasks'. *IEEE Transactions on Computers* **55**(12), 1509–1522.

Shin, D., J. Kim, and S. Lee: 2001, 'Intra-task voltage scheduling for low-energy hard real-time applications'. *IEEE Design & Test of Computers* **18**(2).

Shin, Y. and K. Choi: 1999, 'Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems'. In: *Design Automation Conference*. pp. 134–139.

Stankovic, J., M. Spuri, K. Ramamritham, and G. Buttazo: 1998, *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publisher.

Weisstein, E.: 1999, *The CRC Concise Encyclopedia of Mathematics*. CRC Press. ISBN 0-8493-9640-9.

Xu, R., C. Xi, R. Melhem, and D. Moss: 2004, 'Practical PACE for embedded systems'. In: *Proceedings of the 4th ACM international conference on Embedded software (EMSOFT'04)*. New York, NY, USA, pp. 54–63, ACM Press.

Yao, F.: 2003, 'Complexity of the Yao Demers Shenker Algorithm'. Private communication.

Yao, F., A. Demers, and S. Shenker: 1995, 'A scheduling model for reduced CPU energy'. In: *Proceedings of lEEE Annual Foundations of Computer Science*. pp. 374–382.

Yun, H.-S. and J. Kim: 2003, 'On energy-optimal voltage scheduling for fixed-priority hard real-time systems'. *ACM Transactions on Embedded Computing Systems* **2**(3), 393–430.

Zhang, F. and S. Chanson: 2002, 'Processor Voltage Scheduling for Real-Time Tasks with Non-Preemptible Sections'. In: *23th Real-Time Systems Symposium*. pp. 235–245.

Appendix

A. Appendix: Association of the longest increasing sequences

This appendix is devoted to the proof that the random variables S_i are associated. This property is used in the proof of Lemma 7. For definition and properties of associated variables, we refer to (Barlow and Proschan, 1981).

Actually, a more general result will be proven : If $(x_1, y_1), \ldots, (x_n, y_n)$ are random points in $\mathbb{R}_+^2$, then the corresponding longest increasing sequences $S_1 \ldots, S_n$ are associated.

The proof holds by first discretizing the space $\mathbb{R}_+^2$. Let us consider the set $E_K = \{0, \ldots, K\} \times \{0, \ldots, K\}$. At each point $(i, j) \in E_K$, corresponds a binary random variable $X_{i,j}$. In the following, we assume that the variables $X_{i,j}$ are independent. The meaning of $X_{i,j}$ is the presence (or not) of a point at position (i, j).

Now, $S_{i,j}^K$ is the length of the longest increasing sequence of points, starting at position (i, j) (whether there is a point at position (i, j) or not).

Is it clear that each random variable $S_{i,j}^K$ is an increasing function of $(X_{i,j})_{(i,j)\in E_K}$. Since $(X_{i,j})_{(i,j)\in E_K}$ are independent, then $(S_{i,j}^K)_{(i,j)\in E_K}$ are associated by Theorem 2.2 and Property P_3 , page 30, in (Barlow and Proschan, 1981).

Now, if we merely consider those $S_{i,j}^K$ that actually correspond to the presence of points, this subset is still associated by Property P_1, page 30, in (Barlow and Proschan, 1981). When the number of points happens to be n, those n variables $S_{i,j}^K$ are associated.

Now, to get back to the original continuous case, we let K go to infinity and scale E_K properly, so that the positions (i, j) become dense in the set $\{0, \max_i x_i\} \times \{0, \max_i y_i\}$.