AMCA: A linear algorithm for real-time scheduling with optimal energy use.

Bruno Gaujal, Nicolas Navet, Cormac Walsh

Laboratoire de l'Informatique du Parallélisme

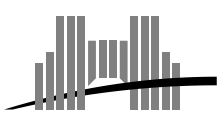
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

A linear algorithm for real-time scheduling with optimal energy use

Bruno Gaujal [1] ,
Nicolas Navet [2] ,
Cormac Walsh [3]

July, 2003

Research Report N° 2003-38



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

A linear algorithm for real-time scheduling with optimal energy use

Bruno Gaujal [4] , Nicolas Navet [5] , Cormac Walsh [6]

July, 2003

Abstract

We present an algorithm for scheduling a set of non-recurrent tasks (or jobs) with real-time constraints so as to minimize the total energy consumption on a dynamically variable voltage processor. Our algorithm runs in linear time and is thus an improvement over the classical algorithm of Yao et al. [14]. It was made possible by considering the problem as a shortest path problem. We also propose an algorithm for the case where the processor possesses only a limited number of clock frequencies. We extend this algorithm to provide the minimum number of speed changes, which is important when the speed switching overhead cannot be neglected. All our algorithms are linear in the number of tasks if the arrivals and deadlines are sorted and need $O(N \log N)$ time otherwise and these complexities are shown optimal. We extend the results to fluid tasks and non-convex cost functions. An interesting by-product of this study is that it furnishes a linear time feasibility test for a set of non-recurrent tasks scheduled under EDF. This is an improvement over the classical one presented in [13].

Keywords: Real-Time Systems, Low-Power, Scheduling, Dynamic Voltage Scaling.

Résumé

Nous proposons un algorithme pour ordonnancer un ensemble de tâches non-récurrentes (ou "jobs") soumises à des contraintes temps réel de façon à minimiser la consommation en énergie sur un processeur dont la tension d'alimentation peut-être modifiée en-ligne ("dynamic voltage scaling"). Cet algorithme est linéaire en le nombre de tâches, ce qui constitue une amélioration par rapport à l'algorithme classique de Yao et al. [14]. Cette amélioration a été possible en considérant ce problème d'ordonnancement comme un problème de plus court chemin. Nous présentons aussi un algorithme pour le cas où le processeur possède un nombre fini de vitesses. Nous l'étendons ensuite pour minimiser le nombre de changement de vitesses ce qui est important lorsque le coût d'un changement de vitesses ne peut être négligé. Tous les algorithmes sont linéaires en le nombre de tâches si les instants d'arrivées et d'échéances sont triés et en $O(N \log N)$ sinon. Ces complexités sont prouvées optimales. Nous étendons ensuite les résultats au cas des tâches fluides et aux fonctions de coût non-convexes. Une contribution intéressante de cette étude est qu'elle fournit un algorithme linéaire pour tester la faisabilité d'un ensemble de tâches ordonnancées sous EDF, ce qui est une amélioration sur le test classique présenté dans [13].

Mots-clés: Systèmes Temps Réel, Économie d'Énergie, Ordonnancement, Adaptation Dynamique du Voltage.

1 Introduction

Context of the study. To provide more functionality and better performance, embedded systems have increasing need for computation power. This requires the use of high frequency electronic components that consume much electrical power. Currently, battery technology is not progressing sufficiently fast to keep up with demand. All battery operated systems, such as PDAs, laptops and mobile phones, would benefit from a better energy efficiency. Reducing energy consumption will not only lead to a longer operating time but also to a decrease of the weight and the space devoted to the battery.

Existing Work. Amongst hardware and software techniques aimed at reducing energy consumption, supply voltage reduction, and hence reduction of CPU speed, is particularly effective. This is because the power dissipated in CMOS circuits is proportional to the square of the supply voltage. In the last few years, variable voltage processors have become available and much 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 task's characteristics (execution time, arrival date, deadline ...) 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 [12] presented a simple run-time strategy that reduces energy consumption. In [10], Quan and Hu proposed a more complex algorithm that was more efficient in their experiments. When the scheduling is made on top of EDF, Yao et al. in [14] proposed an off-line algorithm for finding the optimal voltage schedule of a set of independent tasks. They also presented some on-line heuristics and gave lower bounds on their efficiency. Other on-line heuristics based on EDF have been proposed, for instance in [5] for the problem of scheduling both periodic and aperiodic requests.

Other directions of research concern the discrepancy between worst-case execution times (WCET) and actual execution times. One class of algorithms, known as "stochastic scheduling" [8, 3, 4] consists of finding a feasible speed schedule that minimizes the expected energy consumption. A second class of techniques [9, 11] is known 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. Among alternative approaches, termed "dynamic reclaiming algorithms", one can cite [1] and [15].

Numerous other studies have been conducted on dynamic voltage scheduling; the reader may refer to [4] for a recent survey.

Goal of the paper. The study published in [14] remains one of the most important in the field because it provides, for independent tasks with deadlines, the schedule that minimizes energy usage while ensuring that deadlines are not missed. The algorithm works by identifying the time interval, termed the critical interval, over which the highest processing speed is required. The lowest admissible frequency is computed for this interval, the tasks belonging to this interval (i.e. arrival date and deadline inside the interval) are then removed and a sub-problem is constructed with the remaining tasks. In this study, the problem of finding the minimal voltage schedule is reduced to a shortest path problem. This enables us to provide linear-time algorithms for minimizing the energy consumption of a set of non-recurrent tasks in the following situations:

- when the processor speed range is continuous (the processor can take an arbitrary frequency over the frequency range),
- when the number of speeds is discrete,
- when the number of speeds is discrete and there is the additional objective of minimizing the number of speed changes. To the best of our knowledge, in our context, this problem has not been addressed before.

Another contribution of the paper is a proof that the optimal solution with a discrete number of speeds is to use the two neighboring frequencies that bound the ideal frequency. This result has been shown in [6] for a single task considered alone but, to the best of our knowledge, this was not known for a global optimization over a set of tasks. We also consider the case of fluid tasks and non-convex cost functions.

Organization of the document. Section 2 describes the system model and studies the problem in the case where the processor has a continuous frequency range. In Section 3, the problem where the processor possesses a finite number of speeds is investigated in two steps: first without minimizing the number of speed changes and then with this additional objective. In Section 4 the algorithm leading to the lowest energy schedule is detailed and its linear complexity is proven. Section 5 is dedicated to fluid tasks and Section 6 to non-convex cost functions.

2 Statement of the problem

We consider a system made of a single processing unit dedicated to execute some work with real-time constraints. The work arrival function $A(t)$ is the amount of work that has arrived up to time t. One denotes by $D(t)$ the amount of work that the server must have executed by time t, the values of this function are induced by the deadlines of the tasks. Functions A and D are non-decreasing by definition and for all t, one has $A(t) \geq D(t)$. With no loss of generality, one can assume that $A(0) = 0$ and $D(0) = 0$. Since A and D are non-decreasing, they are piece-wise continuous.

The tasks of the system are characterized by the set $\{(a_n, s_n, d_n)\}_{n=1\cdots N}$ where the quantities a_n, s_n, d_n respectively denote the arrival time, the size and the deadline of task n. Note that such non-recurrent tasks are sometimes termed "jobs" or even "aperiodic tasks" in the literature. Nevertheless, we point out that the results presented in this paper can also be applied to periodic tasks by computing the schedule for the least common multiple of all tasks periods.

The functions $A(t)$ and $D(t)$ are staircase functions (*i.e.* piece-wise constant, with a finite number of pieces) :

$$A(t) = \sum_{i=1}^{N} s_i \cdot \mathbf{1}_{[a_i<t]},$$

$$D(t) = \sum_{i=1}^{N} s_i \cdot \mathbf{1}_{[d_i\leq t]},$$

Note that the function A is left-continuous and the function D is right-continuous.

The CPU processing speed u can vary in time over a continuous range from 0 to 1 (after a possible re-scaling). The case where the range of speeds is made of a finite number of speeds

$\{v_1 \cdots v_\ell\}$ will be investigated in section 3. The system (A, D) is said *feasible* if when setting $u(t) = 1$ (i.e. using the processor at maximal speed) and when scheduling under EDF no time constraint is violated.

The objective of the study is to choose at each time t the speed $u(t)$ in such a way as to execute all tasks within their deadline constraints while minimizing the total energy consumption between time 0 and time T where T is the time horizon of the problem. The energy consumption of the processor at time t, $e(t)$ is a function of its speed, $u(t)$. In the following we assume that $e(t) = g(u(t))$ where g is an arbitrary increasing convex function[7] over $\mathbb{R}_+$. One can express the problem in mathematical terms:

Problem 1. *Find an integrable function* $u : [0, T] \to \mathbb{R}$ *such that*

$$\int_0^T g(u(s))ds \text{ is minimized,} \tag{1}$$

under the constraints

$$u(t) \geq 0 \quad \forall t \in [0, T], \tag{2}$$

$$\int_0^t u(s)ds \leq A(t) \quad \forall t \in [0, T], \tag{3}$$

$$\int_0^t u(s)ds \geq D(t) \quad \forall t \in [0, T]. \tag{4}$$

Condition (2) says that the speed of the processor must remain non-negative. Condition (3) says that the total work executed by the processor up to time t cannot be larger than the total workload arrived up to that date. Condition (4) says that all the work executed by the processor up to time t (namely $\int_0^t u(s)ds$) exceeds the work that must be done before time t (namely $D(t)$). Note that the problem statement only uses the integral $U(t) \stackrel{\text{def}}{=} \int_0^t u(s)ds$ of u. Therefore, the function u is only defined almost everywhere (a.e.). In the following, we will identify all functions which are equal a.e. .

A function u satisfying the constraints is called an *admissible* solution. An admissible u minimizing $\int_0^T g(u(s))ds$ is called an *optimal* solution.

Actually, in order to take into account the fact that the speed of the processor cannot exceed 1, one comes with a more constrained problem:

Problem 2. *Find an integrable function* $u : [0, T] \to \mathbb{R}_+$ *such that*

$$\int_0^T g(u(s))ds \text{ is minimized,}$$

under the constraints (2), (3), (4) *and*

$$u(t) \leq 1 \quad \forall t \in [0, T]. \tag{5}$$

In the rest of the paper, we focus on the problem to determine the optimal speed. We assume that, at any time, the tasks are scheduled under EDF (where ties are broken arbitrarily). In our context, the EDF policy ensures the optimality from a schedulability point of view (see [7] quoted in [13]).

[7] with the CMOS technology, typically, $e(t) \approx \alpha C u(t)^{3/\gamma - 1}$, where $0 \leq \gamma \leq 3, \alpha \leq 0, C \geq 0$, see [4] for more details.

2.1 Main result

In this section, we show several characterizations of the speed functions u that are solutions to Problem 1.

Lemma 1. *If the function g is strictly convex, then the optimal solution of problem 1 is unique (up to a set of measure 0).*

Proof. Let us consider the set of all integrable functions satisfying the constraints (2), (3) and (4). This set is obviously convex. Assume that two functions u_1 and u_2, different over a set S of positive measure, both minimize the energy: $\int_0^T g(u_1(s))ds = \int_0^T g(u_2(s))ds$. Then for all $0 < \alpha < 1$ and all $t \in S$, $g(\alpha u_1(t) + (1-\alpha)u_2(t)) < \alpha g(u_1(t)) + (1-\alpha)g(u_2(t))$ by strict convexity of g. Therefore,

$$\begin{aligned}\int_0^T g(\alpha u_1 + (1-\alpha)u_2)ds &< \int_0^T \alpha g(u_1)ds \\ + \int_0^T (1-\alpha)g(u_2)ds &= \int_0^T g(u_1)ds.\end{aligned}$$

This clearly contradicts the optimality of u_1. □

Theorem 1. *If u^* is the optimal solution of problem 1 where g is strictly convex and non-decreasing, then u^* is also an optimal solution of problem 1 for any other non-decreasing convex function.*

Proof. We first consider the case when g is strictly convex. Consider the problem of minimizing

$$\int_{t_0}^{t_1} g(u(t))dt$$

under $U(t_0) = U_0$ and $U(t_1) = U_1$ in the absence of any other constraints. This is an easy problem in the Calculus of Variations. The solution given by Euler's formula is $d/dt\, g'(u) = 0$, which implies that u is constant. Thus, if $(t_0, U^*(t_0))$ and $(t_1, U^*(t_1))$ are two points on the optimal path U^*, then U^* has constant slope between these two points if this is feasible. We conclude that U^* only changes slope at the arrival times $\{a_n\}_{1\le n\le N}$ or deadline times $\{d_n\}_{1\le n\le N}$. Moreover, when the slope of U^* decreases we must have $U^*(t) = D(t)$ and $t = d_n$ for some $n \in \{1, \ldots, N\}$. Otherwise, in the neighborhood of t, U^* should be a straight line if it were feasible. Likewise, when the slope of U^* increases we must have $U^*(t) = A(t)$ and $t = a_n$ for some $n \in \{1, \ldots, N\}$.

We will show that these properties, together with $U^*(0) = 0$ and $U^*(T) = U_T$, completely determine U^*. Suppose that there are two functions U_1 and U_2 meeting the constraints with the properties above such that $U_1(0) = U_2(0) = 0$. Let τ be the first time that U_1 and U_2 differ. Then the right derivative of U_1 (say) at τ is strictly greater than that of U_2. Therefore U_1 is strictly greater than U_2 at the next event τ_1, be it arrival or deadline. We can have neither $U_1(\tau_1) = A(\tau_1)$ nor $U_2(\tau_1) = D(\tau_1)$. Therefore the slope of U_1 cannot decrease at τ_1 and that of U_2 cannot increase. Proceeding by induction, we get that $U_1(t) > U_2(t)$ for all $t > \tau$. Thus $U_1(T)$ and $U_2(T)$ must differ. □

In the following, the optimal solution u^* of Problem 1 provided by this theorem will be called "the solution of Problem 1". As it will be seen in Section 3, when g is not strictly convex there may be more than one optimal solution.

Note that $g(x) = \sqrt{1+x^2}$ is strictly convex and increasing, and that $\int_0^T \sqrt{1+u^2(s)}ds$ is the length of the curve of the function $U(t) \stackrel{\text{def}}{=} \int_0^t u(s)ds$ from $t = 0$ to $t = T$. Hence, the optimal solution can be interpreted in many ways. One way to see this is the following. Consider a road limited by two concrete walls (the functions A and D resp.). Find the shortest path from the beginning to the end. This gives $U^* \stackrel{\text{def}}{=} \int_0^t u^*(s)ds$.

Corollary 1. *The optimal solution u^* of problem 1 satisfies the following inequality*[8]*:* $\sup_{0\le t\le T} u^*(t) \le \sup_{0\le t\le T} u(t)$*, for all functions u satisfying constraints (2), (3) and (4).*

Proof. Consider the function $g_n(x) = x^n$. The function g_n is increasing and convex over $[0, T]$. By applying Theorem 1, the optimal solution u^* of Problem 1 using $g = g_n$ does not depend on n. Now consider the limit when n goes to infinity: over the interval $[0, T]$, for all functions u,

$$\lim_{n\to\infty} \left(\int_0^T g_n(u(s))ds \right)^{1/n} = \sup_{0\le t\le T} u(t).$$

This shows that $\sup_{0\le t\le T} u^*(t) \le \sup_{0\le t\le T} u(t)$. □

Corollary 2. *If the system (A, D) is feasible, then the optimal solution u^* of problem 1 satisfies $u^*(t) \le 1$, and therefore Problems 1 and 2 are equivalent.*

Proof. Let us consider the case where the system (A, D) is feasible. This means that their exists a solution u to the set of constraints of Problem 2. This solution is also a solution to the set of constraints of Problem 1. Using Corollary 1, the optimal solution of Problem 1 satisfies $\sup_{0\le t\le T} u^*(t) \le \sup_{0\le t\le T} u(t) \le 1$. Therefore u^* is also an optimal solution to Problem 2. □

An example illustrating Problem (1) is given in Figure 1.

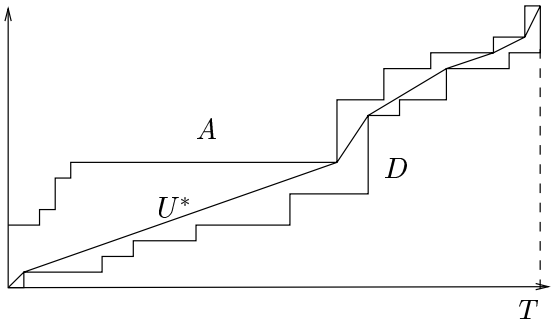


Figure 1: The function U^* is the shortest path from point 0 to T.

Remark 1. *Conversely, if the system is not feasible then the optimal solution of Problem 1 will not satisfies $u^* \le 1$. This provides a feasibility test for a set of non-recurrent independent tasks under EDF.*

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

Actually more precise results can be stated. Since u^* is piece-wise constant, one may only focus on the discontinuity points of u^*. We denote those points by $P_1, \cdots, P_n$. They either belong to the graph of A or to the graph of D. The first point is $P_1 = (0, 0)$ and belongs to both A and D. This sequence of points can be split into sub-sequences of consecutive points belonging to A or to D.

Corollary 3. *Consider a sub-sequence $P_r, \cdots, P_s$ of discontinuity points of u^* such that $P_r \in A, P_s \in A$ and all the other points in between belong to D. Then between points P_r and P_s, U^* is the upper concave envelope of the points $P_r, \cdots, P_s$.*

Dually, consider a sub-sequence $P_i, \cdots, P_j$ of discontinuity points of u^* such that $P_i \in D, P_j \in D$ while all the other points in between belong to A. Then between points P_i and P_j, U^* is the lower convex envelope of the points $P_i, \cdots, P_j$.

Proof. (Sketch) This is a direct consequence of the fact that U^* is the shortest path going through the points $P_1, \cdots, P_n$. □

The function u^* can be computed according to the method described in [14] which is based on the computation of critical intervals. A straightforward implementation of their algorithm requires $O(N^2)$ time where N is the number of tasks. The authors claim, without more details, that using "a suitable data structure such as the segment tree", the running time can be reduced to $O(N \log^2(N))$. We actually do not know how to obtain such a low complexity implementation with their algorithm. An interesting consequence of the construction of u^* as given in [14] is that there exists a schedule with minimal energy consumption such that each task is executed at a constant speed. This is not obvious with our approach. However, as seen in the following sections, our approach has other important advantages. In particular, we will see in section 4 that there exists a linear time algorithm to compute u^*.

3 Finite number of speeds

We now consider the case where the clock frequency of the processor can only take a finite number of values $v_1 \leq \cdots \leq v_\ell$. As explained in [4], in practice this is necessarily the case with today's technology.

Problem 3. *Find an integrable function $z : [0, T] \to \mathbb{R}_+$ such that*

$$\int_0^T g(z(s))ds \text{ is minimized,}$$

under Constraints (3) and (4) and the additional constraint

$$z(t) \quad \in \quad \{v_1, \cdots, v_\ell\}. \tag{6}$$

Let $u^*(t)$ be the solution of Problem 1. We assume that $v_1 \leq u^*(t) \leq v_\ell$ for all $0 \leq t \leq T$, so that the range of speeds which are available cover the speeds needed by the processor. This assumption will be satisfied in the typical situation where $v_1 = 0$ (the processor can idle) and $v_\ell = 1$ (the processor can use its maximal speed), and where the set of tasks is feasible.

We now describe how to construct an optimal solution z^* to Problem 3.

Partition $[0, T]$ into contiguous intervals such that the boundary between intervals are the discontinuity points of U^*, there will be $M < 2N$ intervals. On one such interval, say

$I_k = [b_k, b_{k+1})$, u^* is constant as seen in the proof of Theorem 1, equal to, say, u_k^* that falls in between two possible speeds for the processor, v_i and v_{i+1}. Let α_k be defined by $u_k^* = \alpha_k v_i + (1-\alpha_k)v_{i+1}$ Now, let us construct a function over I_k: $z^*(t) = v_i$ over $[b_k, c_k)$ and $z^*(t) = v_{i+1}$ over $[c_k, b_{k+1})$ where $c_k = (1-\alpha_k)b_k + \alpha_k b_{k+1}$. It is clear that the function $z^*(t)$ is an admissible solution for Problem 3 since it satisfies all the constraints. Furthermore, as shown in the following theorem, it is an optimal solution.

Theorem 2. *Under the foregoing assumptions, the function $z^*(t)$ is an optimal solution to Problem 3.*

Proof. Using the assumption on the range of the v_i's, for any u such that $v_1 \leq u < v_\ell$, there exist $i(u)$ such that $v_{i(u)} \leq u < v_{i(u)+1}$. We then introduce the coefficient α_u such that $u = \alpha_u v_{i(u)} + (1-\alpha_u)v_{i(u)+1}$ and consider the real function $\tilde{g}(u) \stackrel{\text{def}}{=} \alpha_u g(v_{i(u)}) + (1-\alpha_u)g(v_{i(u)+1})$.

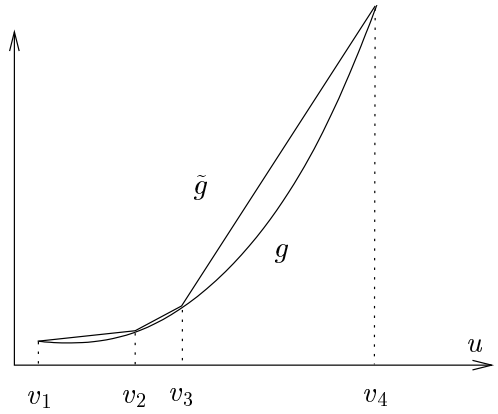


Figure 2: The functions g and its linear interpolation $\tilde{g}$.

First, note that $\tilde{g}$ is the linear interpolation of g over the points $v_1, \cdots v_\ell$. Since g is convex and non-decreasing, $\tilde{g}$ is also convex and non-decreasing.

The second part of the proof consists in showing that

$$\int_0^T g(z^*(s))ds = \int_0^T \tilde{g}(u^*(s))ds. \tag{7}$$

Indeed, using the definition of z^*,

$$\int_0^T g(z^*(s))ds = \sum_{k=0}^{M} \int_{I_k} g(z^*(s))ds = \sum_{k=0}^{M} \left(\int_{b_k}^{c_k} g(v_i)ds + \int_{c_k}^{b_{k+1}} g(v_{i+1})ds \right)$$

$$= \sum_{k=0}^{M} ((c_k - b_k)g(v_i) + (b_{k+1} - c_k)g(v_{i+1})) = \sum_{k=0}^{M} (b_{k+1} - b_k)(\alpha_k g(v_i) + (1-\alpha_k)g(v_{i+1}))$$

$$= \sum_{k=0}^{M} (b_{k+1} - b_k)\tilde{g}(u_k^*) = \int_0^T \tilde{g}(u^*(s))ds.$$

Now, let z be any admissible solution of Problem 3. Therefore, $z(t) \in \{v_1 \cdots, v_\ell\}$. Since the function $\tilde{g}$ coincides with g over $\{v_1 \cdots, v_\ell\}$, then one has $\int_0^T g(z(s))ds = \int_0^T \tilde{g}(z(s))ds$.

Now using the fact that $\tilde{g}$ is increasing and convex, $\int_0^T \tilde{g}(z(s))ds \geq \int_0^T \tilde{g}(u^*(s))ds$. Finally, Equality (7) shows that $\int_0^T g(z(s))ds \geq \int_0^T g(z^*(s))ds$. This means that the energy use of any admissible solution z is larger than the energy use of z^*. □

Finally, note that the construction of z^* can be done in linear time, once the function u^* is given. The construction of z^* is illustrated in Figure 3.

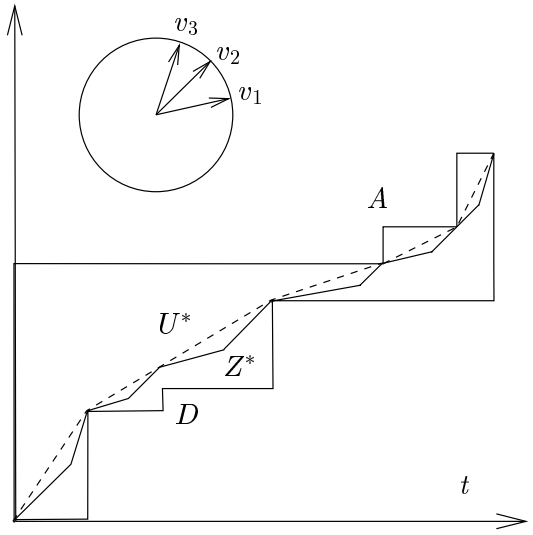


Figure 3: The function Z^* is the integral of an optimal solution z^* when using 3 speeds, v_1, v_2 and v_3.

Remark 2. *The computation time of Z^* given U^* is linear in the number of tasks.*

It would be interesting to study the difference in energy consumption between the continuous case where the speed can range over the whole interval $[0, 1]$ and the case where it can take only finitely many values. By uniform convergence arguments, it should be obvious that they coincide in the limit when the maximal gap between two consecutive admissible speeds goes to zero.

3.1 Minimal number of speed changes

Because the modified cost function $\tilde{g}$ is not strictly convex, there will be many different optimal solutions to Problem 3. Amongst them will be z^*. However, it may be possible to find an optimal solution with fewer speed changes than z^*. We now present an algorithm for doing this.

The main idea of the construction is to switch between speeds only when absolutely necessary. Suppose we are at a point $(\tau, M(\tau))$ in an interval I in which z^* uses only two speeds v_h and v_{h+1}. If the current processor speed is v_h, then the latest time we can switch to speed v_{h+1} while still meeting the constraints is $\sup\{t : M(\tau) + (t - \tau)v_h \geq \tilde{D}(t)\}$, where

$$\tilde{D}(t) := \max_{d_i \geq t} [D(d_i) - (d_i - t)v_{h+1}].$$

Similarly, if the current processor speed is v_{h+1}, then the latest time we can switch to speed v_h and still be guaranteed not to run of out work is $\sup\{t : M(\tau)+(t-\tau)v_{h+1} \leq \tilde{A}(t)\}$, where

$$\tilde{A}(t) := \min_{a_i \geq t}[A(a_i) - (a_i - t)v_h].$$

The "latest switching" algorithm in the interval I consists of alternating between the speeds v_h and v_{h+1} in the above manner. We use this algorithm to construct an optimal function with a minimum number of speed changes:

1. Partition $[0, T]$ into intervals $I_1, \cdots, I_k$ such that in each I_i, the function z^* only uses the same two speeds, say v_h, v_{h+1}. We also require that the partition is the coarsest possible in the sense that the pair of speeds used by z^* in neighboring intervals are different.

2. In each interval $I_i = [a_i, b_i]$, calculate the following two functions $\underline{m}_i$ and $\overline{m}_i$ using the "latest switching" algorithm above. The first starts at $(a_i, z^*(a_i))$ with initial speed v_h and finishes at $(b_i, z^*(b_i))$. The second has the same start and finish points but has initial speed v_{h+1}. Record the terminal speed of both functions and their number of speed changes.

3. Initialize $\overline{m}$ and $\underline{m}$ to be empty. Go through the intervals in reverse order, applying the following recursive procedure. At each stage, append to $\overline{m}_i$ either $\overline{m}$ or $\underline{m}$ so as to minimize the total number of speed changes, including the possible speed change at the interface. All the necessary information was calculated during the previous step. The resulting function is the new $\overline{m}$. Similarly, we append either the old $\overline{m}$ or $\underline{m}$ to $\underline{m}_i$ to form the new $\underline{m}$.

4. We end up with two functions $\overline{m}$ and $\underline{m}$. Choose the one with fewer speed changes and call it m^*.

Remark 3. *It is straightforward that the computation time of m^* given z^* is linear in the number of tasks.*

Theorem 3. *The function m^* constructed by this algorithm is an optimal solution to Problem 3 and any other optimal solution has at least as many speed changes.*

Proof. First note that M^* and Z^* agree at the end points and use the same two speeds in each interval, I_i. Therefore, they use each speed for the same amount of time and hence use the same amount of energy. This shows that m^* is an optimal solution to problem 3.

Now let m be any feasible function that uses the speeds $\{v_h : 1 \leq h \leq l\}$ is the same proportions as does z^*. Let the maximum speed of u^* lie between v_h and v_{h+1} and consider the set of intervals $\{I_k : k \in K\}$ in the partition where u^* lies between v_h and v_{h+1}. Let $I_i = [a_i, b_i]$ be one of these intervals. In the neighboring intervals I_{i-1} and I_{i+1}, we have $z^* < v_h$. Therefore, $Z^*(a_i) = A(a_i)$ and $Z^*(b_i) = D(b_i)$. Thus

$$\int_{a_i}^{b_i} m\, dt \geq Z^*(b_i) - Z^*(a_i). \tag{8}$$

Let τ_i' and τ_i be, respectively, the amount of time m and z^* spend at speed v_{h+1} in the interval I_i. Since the total time spend at speed v_{h+1} is the same for both m and z^*, if $\tau_i' > \tau_i$ for

some $i \in K$ then to compensate there must be some $j \in K$ such that $\tau'_j < \tau_j$. However, since $\int_{a_i}^{b_i} m\,dt \leq \tau'_i v_{h+1} + (b_j - a_j - \tau'_j)v_h$, this would contradict (8). We conclude that $\tau'_i = \tau_i$ for all $i \in K$ and, moreover, that in each of the intervals $\{I_k : k \in K\}$ the function m uses the same two speeds as z^* for the same amount of time. By now removing these intervals and applying an inductive argument, we may extend the same conclusion to all the intervals in the partition. This provides a justification for considering each interval separately.

If m is not equal to the function m^* constructed above, then at some point m switches between speeds earlier than necessary. If delaying both this switch and the following switch in the opposite direction, then we obtain another optimal solution to Problem (3) which has the same or fewer speed changes as m. By doing this each time m switches too early, and choosing the delays appropriately, m may be transformed into m^*. Therefore m^* has the minimum possible number of speed changes. □

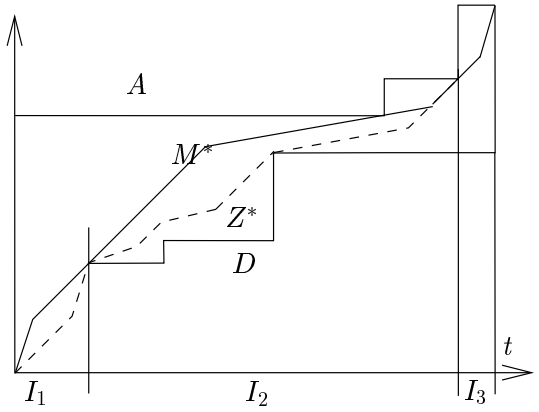


Figure 4: An optimal solution with a minimal number of speed changes.

Figure 4 shows both integrals Z^* and M^* in the example displayed in 3. In this case, m^* has 4 speed changes while z^* has 10 speed changes. The function m^* was built by using the latest switching algorithm. In interval I_3, the two speeds are v_2 and v_3 and m^* uses v_2 first. In I_2, the two speeds are v_1 and v_2. The best solution uses v_2 first. Finally in I_1, the two speeds are v_2 and v_3 and the best is to start with v_3.

4 A linear algorithm to compute u^*

While Theorem 1 characterizes the function u^*, it is not constructive. This section shows how to construct u^*. The function A (resp. D) is given under the form of an ordered list $L_A := [(a_1, A(a_1)), \cdots (a_N, A(a_N))]$ (resp. $L_D := [(d_1, D(d_1)), \cdots, (d_N, D(d_N))]$ with $a_1 < \cdots < a_N$ (resp. $d_1 < \cdots < d_N$).

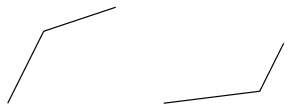
We propose an algorithm that constructs the function $U^*(t)$ (as well as $u^*(t)$) under the form of an ordered list $(x_1, y_1), \cdots, (x_K, y_K)$ with $x_1 < \cdots < x_K$. The function U^* being the linear interpolation between those points. This algorithm is similar to the linear time algorithm computing the convex hull of n ordered points in the plane (see for example [2]).

The main idea of the algorithm is to construct two piece-wise affine functions V, W, inductively, by introducing the points of A and D one by one. Both functions start with the

same initial value at the initial point : $V(x_0) = W(x_0)$. The function U^* and V or W share a common prefix, determined by the algorithm.

step 0 Merge the lists L_A and L_D into a single list L ordered by the first coordinate.

step 1 Set $x_0 := (0,0)$, $V := [(0,0)]$, $W := [(0,0)]$.



hat-angle cup-angle

Figure 5: A hat-angle and a cup-angle

step 2 Sweep the list L. Here is the invariant of the algorithm after n steps (made of i points in A and j points in D). $V = [(V_0^1, V_0^2), \cdots (V_k^1, V_k^2)]$ is the lower convex hull of the function A from point x_0 to $(d_i, A(d_i))$ and $W = [(W_0^1, W_0^2), \cdots, (W_m^1, W_m^2)]$ is the upper concave hull of the function D from point x_0 to $(d_j, D(d_j))$.

- If the next point in L belongs to A (*i.e.* it is $(d_{i+1}, A(d_{i+1}))$),
  1. add it in the last position $(k+1)$ in the list $V := V \cdot (d_{i+1}, D(d_{i+1}))$.
  2. Update the list V by removing the points starting from (V_k^1, V_k^2) and going backwards as long they form a hat angle.
  3. If all the points are removed (the first point (V_0^1, V_0^2) cannot be removed), update everything as follows:
  - $\ell := 0$, while V is below W at point $W_{\ell+1}$ do $\ell := \ell + 1$ od.
  -$U^* := U^* \cdot [(W_0^1 W_0^2) \cdots (W_\ell^1, W_\ell^2)]$;
  -$W := [(W_\ell^1, W_\ell^2), \cdots, (W_m^1, W_m^2)]$;
  -$V := [(W_\ell^1, W_\ell^2), (V_{k+1}^1, V_{k+1}^2)]$;
  -$x_0 := (W_\ell^1, W_\ell^2)$.
- If the next point belongs to D (*i.e.* $(d_{j+1}, D(d_{j+1}))$), do the same as above by switching the role of V and W, replacing the hat-angle test with a cup-angle test and testing if W is above V instead of V below W.

step 3 Once the last point in L has been swept, update for the last time the list U^* by concatenating U^* and W: $U^* := U^* \cdot W$.

Notice that the algorithm constructs U^* rather than u^*. However, since U^* is piece-wise affine, it is rather easy to retrieve u^* from U^*.

A run of the algorithm is detailed in Figures 6,7,8. Figure 6 gives the current position where V and W have been constructed up to the current point. As displayed in Figure 7, the next point W_4 belongs to D, so that we update W. The angle at point W_3 is a cup-angle and therefore it is removed from the list. The point W_2 does not form a cup-angle in the new W function so that the removing stops and the new function W is constructed. In Figure 8, we

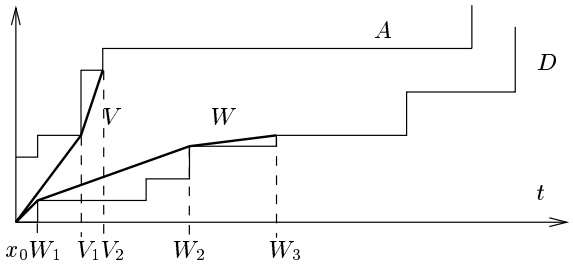


Figure 6: The current position.

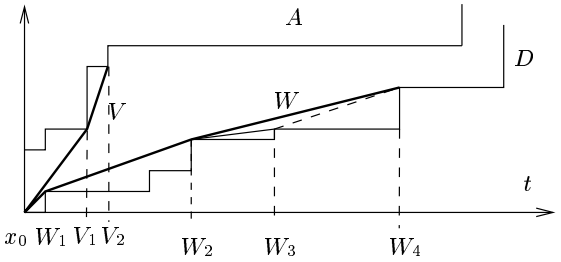


Figure 7: A simple case where only W is updated.

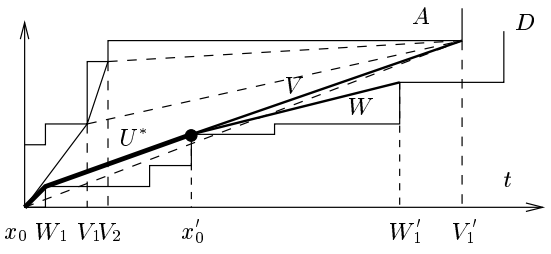


Figure 8: A case where x_0 is updated.

add yet another point, V_3. We update V. Point V_2 forms a hat-angle and is removed. Point V_1 forms a hat-angle and is removed. Finally, all the old points in V were removed, and we have to check whether V remains above W. This is not the case. Point W_1 is above V. Point W_2 is also above V. Finally, Point W_3 is below V. This means that we update the starting points of both V and W to the new starting point $x_0' = W_2$. The new V is made of two points x_0' and V_4, while the new W is also made of two points x_0' and W_3. As for the function U^*, it has been constructed from 0 to the new starting point, x_0' (thicker line in the figure).

Theorem 4. *The algorithm detailed above constructs the function u^* in time $O(N)$, using a memory size $O(N)$.*

Proof. The fact that the function constructed by the algorithm is actually u^*, the optimal solution of Problem 1 is a direct consequence of the proof of Theorem 1 and Corollary 3.

The proof that the algorithm runs in linear time with linear memory size is similar to the computation of the convex hull of a set of ordered points ([2]). A simple way to see this is to notice that the total time needed to construct V, W and u^* is proportional to the number of changes occurring over the lists V, W and u^*. Since each point in these lists is eliminated at most once, the number of changes is proportional to N. □

Note that if the original data is under the form of the set of tasks rather than staircase functions A and D, then one needs, as the first step, to create the lists L_A and L_D which may take $O(N \log(N))$ time if the tasks are not already sorted. In any case, our algorithm is an improvement over the previously known algorithm given in [14] which requires at least $O(N \log^2 N)$ time.

4.1 Optimal complexity

In the previous section, we have shown that the construction of the function u^* (under the form of an ordered list of its points of discontinuity, called L^* in the following) requires $O(N)$ elementary operations (comparisons, additions and multiplications) when the data lists L_A and L_D are already sorted.

This complexity is optimal in the sense that all algorithms must at least examine all the points in the data lists to construct u^*.

When the data lists L_A and L_D are not already sorted, then our algorithms needs a pre-processing phase to sort them before actually construct u^*. The total complexity jumps to $O(N \log N)$.

Next, we show that the computation of the list L^* is more complex than sorting $L_A \cup L_D$. Consider the following problem:
Input: the set of the arrivals (unsorted): $\{a_i, i = 1 \cdots N\}$ and deadlines (unsorted) $\{d_i, i = 1 \cdots N\}$.
Output: the sorted list L^*.

If the size of the jumps of the cumulative functions A and D are chosen appropriately, then the list L^* will contain all the discontinuity points of A and D, so that it is actually equivalent to a sorted list of the set $\{a_i, i = 1 \cdots N\} \cup \{d_i, i = 1 \cdots N\}$.

Here is a way to choose the jumps of both A and D. The choice is made iteratively. Assume that the first $i-1$ discontinuity points have been constructed already. We now look at the i-th point (p_i), which may be either a discontinuity of A or of D. There are two cases: if the previous point belongs to D (say $(d_k, D(d_k))$, then choose the height of the current point

(regardless of the fact that it belongs to A or D), in the interval $[D(d_k), D(d_k)+(p_i-d_k)u^*(d_k)]$. If the previous point belongs to A (say $(a_j, A(a_j))$, then choose the height of the current point (regardless of the fact that it belongs to A or D), above $(p_i - a_j)u^*(a_j)]$.

With these choices of the functions A and D, we make sure that all the discontiniuity points of A and D are also discontinuity points of u^*, so that they will all be listed in L^*.

The arithmetic complexity of the computation of U^* is $O(N \log(N))$ operations (counting additions, multiplications and comparisons as the only elementary operations). This also allows us to sort the initial list $L_A \cup L_D$ in the same amount of time. Up to our knowledge, no algorithm sorting a list of numbers with arithmetic complexity (number of additions, multiplications and comparisons, regardless of the size of the numbers) lower than $O(N \log(N))$ has been found so far. This provides strong evidence that our algorithm has the lowest possible arithmetic complexity.

5 Extension 1: fluid tasks

In this section, we generalize the problem for arbitrary functions A and D, without assuming that they are staircase functions with a finite number of discontinuities. This may model infinetesimal tasks modeled by a fluid arrival process (with a fluid deadline as well).

In this section, we consider two functions A and D satisfying the following properties:

(F_1) A is a non-decreasing left-continuous function, with right and left derivatives in $\mathbb{R} \cup \{-\infty, +\infty\}$ and $A(0) = 0$.

(F_2) D is a non-decreasing right-continuous function, with right and left derivatives in $\mathbb{R} \cup \{-\infty, +\infty\}$, $D(0) = 0$, and $D \geq A$.

Problems 1 and 2 remain unchanged: find an integrable u that minimize the energy spent between time 0 and time T, while satisfying the constraints (3) and (4) (and (5) resp.).

The solution is also the same: the optimal speed allocation for the processor is given by the shortest path between A and D from point $(0, 0)$ to point $(T, D(T))$, since the proof of Theorem 1 also works for arbitrary functions A and D. However, this time, the optimal solution U^* is not necessarily piecewise affine as shown in the example of Figure 9.

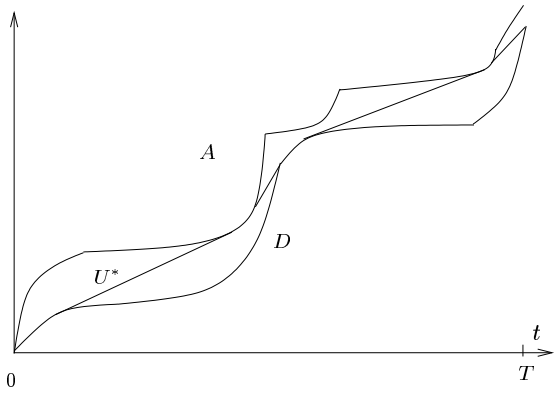


Figure 9: the optimal solution with arbitrary function A and D.

The computational issues becomes here a real concern here because the functions A and D can be arbitrarily difficult to code in a computer program. However, if both A and D are piecewise polynomials (of degree k), then the computation of U^* only involves solving polynomial equations of degree k. This can be done with an arbitrary precision using symbolic computation tools based on Schur polynomials and Grobner basis. Here, the complexity of the algorithm is at least exponential in k.

5.1 Finite number of speeds

As for Problem 3, this requires some changes since taking some precautions becomes necessary. We need to add technical assumptions on the functions A and D, to avoid cases where there are no admissible solutions. One way to make sure that the set of admissible solutions is not empty is by adding the following assumptions on A and D.

(F_3) $\exists\ \delta > 0, \quad \forall\ 0 \leq a \leq b \leq T, \quad \int_a^b A(s) - D(s)ds \geq \delta(b-a).$

(F_4) there exists a finite number of points x between 0 and T such that $A(x) = D(x)$, and $\forall\ x$ s.t. $A(x) = D(x), \exists\ v, w \in \{v_1, \cdots v_\ell\}$ s.t. $\frac{dA}{dt_+}(x) \geq v$, $\frac{dD}{dt_+}(x) \leq v$, and $\frac{dA}{dt_-}(x) \leq w$, $\frac{dD}{dt_-}(x) \geq w$.

If assumption (F_4) is not satisfied, it should be obvious that Problem 3 does not have any admissible solution since at time 0 any choice of the initial speed would break one constraint (either 3 or 4). As for assumption (F_3), it adds the fact that whenever A is strictly above D, there is still enough space between A and D, for some admissible solution using a finite number of speeds.

To find the optimal solution to problem 3, one must construct a finite sequence of functions, (y_n) using the following procedure.

Let $x_1 = 0, x_2, \cdots, x_h = T$ be the points where A and D meet. Partition each interval $[x_i, x_{i+1}]$ of length $T_i \stackrel{\text{def}}{=} x_{i+1} - x_i$ into n sub-intervals of the same size (T_i/n). Here is a way to construct the function y_n.

At step k, we construct the function $y_n(t)$ over the kth interval, namely $I_k \stackrel{\text{def}}{=} (x_i + kT_i/n, x_i + kT_i/n + T_i/n]$. There exists h such that

$$v_h T_i/n \leq \int_{x_i+kT_i/n}^{x_i+(k+1)T_i/n} u^*(s)ds \leq v_{h+1} T_i/n.$$

There exists $\alpha_k \in (0, 1]$ such that

$$u_k^* \stackrel{\text{def}}{=} n/T \int_{x_i+kT_i/n}^{x_i+(k+1)T_i/n} u^*(s)ds = \Big(\alpha_k v_h + (1-\alpha_k)v_{h+1}\Big)$$

From that point on, we have two possibilities to define the function y_n over the interval $I_k = (kT_i/n, kT_i/n + T_i/n]$. At least one of them will be admissible when n becomes large enough.

First alternative: $y_n(t) = v_h$ over the interval $I_k = (x_i + kT_i/n, x_i + \alpha_k T_i/n + (1 - \alpha_k)(kT_i/n + T_i/n)]$ and $y_n(t) = v_{h+1}$ over the interval $(x_i + \alpha_k T_i/n + (1-\alpha_k)(kT_i/n + T_i/n), x_i + kT_i/n + T_i/n]$.

Second alternative: $y_n(t) = v_{h+1}$ over the interval $(x_i + kT_i/n, x_i + \alpha_k(kT_i/n + T_i/n) + (1-\alpha_k)T_i/n]$ and $y_n(t) = v_h$ over the interval $(x_i + \alpha_k(kT_i/n + T_i/n) + (1-\alpha_k)T_i/n, x_i + kT_i/n + T_i/n]$.

Note that because of assumption F_4, this is locally admissible at the extreme points of the intervals.

Theorem 5. *The function $y^* = y_n$ is an optimal solution of Problem 3 for the smallest n such that y_n is admissible.*

Proof. First note that the integral of y_n converges to U^* piecewise when n goes to infinity. Using assumption F_3, this implies that the function y_n is admissible if n is finite but large enough. Second, using the assumption on the range of the u_i's, for any $v_1 \leq u < v_\ell$, there exist $i(u)$ such that $v_{i(u)} \leq u < v_{i(u)+1}$. We then define the coefficient α_u such that $u = \alpha_u v_{i(u)} + (1-\alpha_u)v_{i(u)+1}$. We use the real function $\tilde{g}(u)$ as in Section 3.

The next part of the proof consists in showing that

$$\int_0^T g(y^*(s))ds = \int_0^T \tilde{g}(u^*)ds. \tag{9}$$

Using the definition of y^*,

$$\begin{aligned} \int_0^T g(y^*(s))ds &= \sum_{k=0}^{N} \int_{I_k} g(y_n(s))ds \\ &= \sum_{k=0}^{N} \int_{I_k} \tilde{g}(u_k^*)ds. \end{aligned}$$

Since the function $\tilde{g}$ is affine on all the intervals I_k, then $\sum_{k=0}^{N} \int_{I_k} \tilde{g}(u_k^*)ds = \int_0^T \tilde{g}(u^*)ds$.

Here is now the last part of the proof, which resembles the case with discrete tasks. Let u be any admissible solution of problem 3. Therefore, $u(t) \in \{v_1 \cdots, v_\ell\}$. Since the function f coincides with g over $\{v_1 \cdots, v_\ell\}$, then one has $\int_0^T g(u(s))ds = \int_0^T \tilde{g}(u(s))ds$. Now using the fact that f is increasing and convex, $\int_0^T \tilde{g}(u(s))ds \geq \int_0^T \tilde{g}(u^*(s))ds$. Finally, using Equality (9) shows that $\int_0^T g(u(s))ds \geq \int_0^T g(y_n(s))ds$. This means that the cost of any admissible solution is larger than the cost of y_n. □

6 Extension 2: non-convex cost functions

In this section, we keep the fluid functions A and D under the assumptions F_1, F_2, F_3 and F_4. In this section, we also consider the case where the function g, which gives the instantaneous energy consumption is not convex and increasing. This is typically the case when the static power dissipated by the processor is not neglectable. In this case, the typical behavior of g is displayed in Figure 10.

We assume that the function g is semi-continuous but not convex and increasing. For technical reasons, we will further assume that g has a finite number of inflexion points.

In this case, the optimal solution v^* of problem 1 depends on g. Here is a way to construct v^*.

The first step is to construct the convex hull h of g. Since $g(0) = 0$ and $g(x) \geq 0, \forall x \geq 0$, then h is an increasing convex function. Let $\mathcal{C}$ be the set of points where g and h coincide:

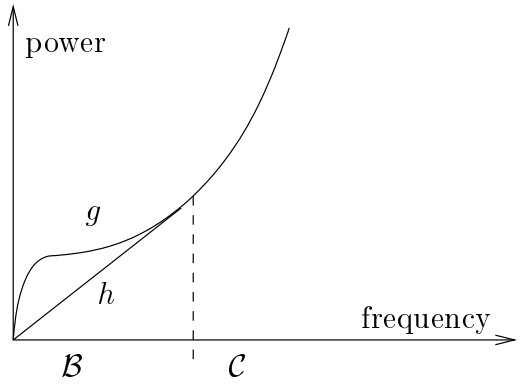


Figure 10: Example of a non-convex energy consumption function g, with its convex hull h, and the sets $\mathcal{C}$ and $\mathcal{B}$.

$\mathcal{C} \stackrel{\text{def}}{=} \{x \in \mathbb{R}_+ \; s.t. \; h(x) = g(x)\}$ and let $\mathcal{B}$ be the complementary set: $\mathcal{B} \stackrel{\text{def}}{=} \{x \in \mathbb{R}_+ \; s.t. \; h(x) \neq g(x)\}$. Using the assumption on the inflexion points of g, the set $\mathcal{B}$ is made of a finite number of intervals. Note that the function h is affine over $\mathcal{B}$. For each $x \in \mathcal{B}$, we define two points in C surrounding x: $\overline{m}(x) \stackrel{\text{def}}{=} \inf\{s \in \mathcal{C} \; s.t. \; s \geq x\}$ and $\underline{m}(x) \stackrel{\text{def}}{=} \sup\{s \in \mathcal{C} s.t. s \leq x\}$.

The second step is to solve problem 1 using h instead of g, as the instantaneous cost. Since h is convex and increasing, we get as before the shortest path U^* between A and D.

The third step is to construct a set of functions, $v_n, n \in \mathbb{N}$ as follows.
If $u^*(t) \in \mathcal{C}$, then $v_n(t) = u^*(t)$.
If $u^*(t) \in \mathcal{B}$, then there exist an interval I, containing t and maximal for inclusion over which $u^* \in [\underline{m}(u^*(t)), \overline{m}(u^*(t))]$. We partition the interval I into n sub-intervals, each of size $|I|/n$. In each such interval, say $[t_1, t_2)$, the average value of u^* over this interval is $\mu \stackrel{\text{def}}{=} \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} u^*(t)dt$ and the coefficient $\alpha \stackrel{\text{def}}{=} \frac{\mu - \underline{m}(\mu)}{\overline{m}(\mu) - \underline{m}(\mu)}$.
Now, $v_n(t) = \underline{m}(\mu)$ over $[t_1, t_1 + (1-\alpha)(t_2 - t_1))$ and $v_n(t) = \overline{m}(\mu)$ over $[t_1 + (1-\alpha)(t_2 - t_1), t_2)$.

The final step is the choice of $v^*(t) = v_n(t) \quad \forall t \in [0, T]$, for some n large enough so that $v^*(t)$ is admissible.

Theorem 6. *The function v^* is the optimal solution to Problem 1.*

Proof. (sketch) The proof is similar to the proof given in Section 5.1. The first thing to notice is that since the intervals used to define the functions v_n get smaller and smaller, their integrals converge point-wise towards $U^*(t)$ when n grows. Therefore since $A > D$, there exists a finite n such that v_n satisfies the constraints 3 and 4 and is admissible.

The second key point in the proof is to notice that since h is affine over $\mathcal{B}$ then $\int_{t_1}^{t_2} h(v_n)dt = h(\mu)(t_2 - t_1)$. By integrating over all the time range, this gives

$$\int_o^T h(v^*)dt = \int_o^T h(u^*)dt. \tag{10}$$

To finish the proof, take u any admissible solution for problem 1.

$$\begin{aligned}
\int_o^T g(u(t))dt &\geq \int_o^T h(u(t))dt, && (11)\\
&\geq \int_o^T h(u^*(t))dt, && (12)\\
&= \int_o^T h(v^*(t))dt, && (13)\\
&= \int_o^T g(v^*(t))dt, && (14)
\end{aligned}$$

where Inequality (11) comes from the fact that $g \geq h$; Inequality (12) comes from the fact that u^* is the optimal solution with the convex cost h; Equality (13) is the same as (10); and Equality (14) comes from the fact that $v^*(t) \in \mathcal{C}$ for all t and the fact that $g = h$ over $\mathcal{C}$. □

6.1 Finite set of speeds

The case where the processor can only take a finite number of speeds $\{v_1, \cdots, v_\ell\}$ is much easier to handle.
First, construct the convex hull h of the finite set of points $\{(v_1, g(v_1)), \cdots, (v_\ell, g(v_\ell))\}$.
Second, remove all the speeds which do not belong to the convex hull from the set of admissible speeds.
Last, solve Problem 3 as in section 3 with the reduced set of speeds. This gives the optimal solution.

7 Conclusion

In this study, we presented a new approach to determine the optimal frequency schedule of a set of independent tasks subject to real-time constraints. The immediate advantage of this proposal is that it can be implemented in linear time (if the functions A and D are given - in $O(N \log(N))$ otherwise) for continuous processor speed range as well as for a discrete number of speeds. In the latter case, we provide an algorithm that ensures the minimum number of speed changes and thus minimizes the speed changing overhead. The results have been extended to fluid tasks and non-convex cost functions.

An interesting contribution of the approach developed in this paper is that it provides a feasibility test under EDF that is less complex than the existing ones [13].

It has been shown that in the context of this study the problem of minimizing the energy consumption is equivalent to a shortest path problem. This observation might possibly lead to some new advances in the field of dynamic voltage scaling.

We are currently investigating the on-line case with probabilistic assumptions on the workload arrival. Two distinct objectives are considered: minimizing the expected energy consumption and minimizing the worst-case energy consumption.

References

[1] H Aydin, Melhem R., D. Mossè, and Mejia-Alvarez P. Dynamic and agressive scheduling techniques for power aware real-time systems. In *Real-Time Systems Symposium*, pages 95–105, 2001.

[2] J.D. Boissonnat and M. Yvinec. *Géométrie Algorithmique*. Ediscience International, 1995.

[3] F. Gruian. 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*, pages 11–16, 2001.

[4] F. Gruian. *Energy-Centric Scheduling for Real-Time Systems*. PhD thesis, Lund Institute of Technology, 2002.

[5] I. Hong, M. Potkonjak, and M.B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *International Conference on Computer Design*, pages 653–656, 1998.

[6] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low Power Electronics and Design*, pages 197–202, 1998.

[7] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical report, University of California, 1955. Report 43.

[8] J.R. Lorch and A.J. Smith. Improving dynamic voltage scaling algorithms with pace. In *ACM SIGMETRICS 2001 Conference*, pages 50–61, 2001.

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

[10] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Design Automation Conference*, pages 828–833, 2001.

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

[12] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Design Automation Conference*, pages 134–139, 1999.

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

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

[15] F. Zhang and S.T. Chanson. Processor voltage scheduling for real-time tasks with non-preemptible sections. In *Real-Time Systems Symposium*, pages 235–245, 2002.