On the Average Complexity of the Processor Demand Analysis for Earliest Deadline Scheduling

Giuseppe Lipari[1], Laurent George[2], Enrico Bini[3], Marko Bertogna[4]

[1] LSV, ENS - Cachan, France ‡
[2] University of Paris-Est, LIGM, France
[3] Lund University, Sweden
[4] University of Modena, Italy

Abstract. Schedulability analysis of a set of sporadic tasks scheduled by EDF on a single processor system is a well known and solved problem: the Processor Demand Analysis is a necessary and sufficient test for EDF with pseudo-polynomial complexity. Over the years, many researchers have tried to find efficient methods for reducing the average-case running time of this test. The problem becomes relevant when doing sensitivity analysis of the worst-case execution times of the tasks: the number of constraints to check is directly linked to the complexity of the analysis. In this paper we describe the problem and present some known facts, with the aim of summarising the state of the art and stimulate research in this direction.

1 Introduction

The *Processor Demand Analysis* is a necessary and sufficient algorithm for testing the schedulability of a set of real-time synchronous periodic or sporadic tasks to be scheduled by the Earliest Deadline First on a single processor. It was first proposed by Baruah et al. [2], and it was later extended to account for more complex task models, shared resources, etc.

The core of the analysis is the computation of the *Demand Bound Function*: the analysis consists in checking that in each interval of time the function does not exceed the length of the interval (more details in Section 2). The problem has been proven to be NP-Hard [2], in the sense that in the worst case it is necessary to analyse the demand bound function over a number of intervals that is exponential in the number of tasks.

Nevertheless, very efficient algorithms have been proposed. A notable example is the QPA algorithm [4], which iterates over the values of the demand bound function in intervals of decreasing length. In average it requires a very small number of steps to assess the schedulability.

‡ The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 246556.

George and Hermant [3] proposed a characterisation of the space of computations times of the tasks (named C-space) that makes the set schedulable. In practice, the constraint that the demand bound function in a certain interval must be less than the length of the interval is interpreted as an inequality where the computation times are unknown. The schedulability test is a set of inequalities that defines a convex polyhedron in the space of the computation times. In general, each different interval produces a different inequality, hence the number of inequalities is exponential. George and Hermant observed that, given a task set, many of the inequalities are redundant, in the sense that they can be safely eliminated without introducing any new solution. In particular, as we will see in Section 3, the number of necessary inequalities is much lower than the total number of intervals that are necessary in theory.

In this paper we further investigate the method proposed by George and Hermant, with the goal of trying to gain additional insight into the complexity of the problem of testing the schedulability of EDF-scheduled task systems.

2 Background

A real-time task τ_i is an infinite sequence of jobs, $J_{i,k}(a_{i,k}, c_{i,k}, d_{i,k})$, where $a_{i,k}$ is the job's arrival time, $c_{i,k}$ is its computation time and $d_{i,k}$ is its absolute deadline. The goal of a real-time scheduling algorithm is to execute the sequence of incoming jobs on the hardware machine (in our case a single processors) so that each job $J_{i,k}$ executes exactly $c_{i,k}$ units of execution time in its execution windows $[a_{i,k}, d_{i,k}]$. The Earliest Deadline First scheduling algorithm selects the active job with the earliest absolute deadline.

A sporadic task τ_i is characterised by a triplet (C_i, D_i, T_i), where C_i is the worst-case computation time, D_i is the relative deadline and T_i is the period, or minimum inter-arrival time. For a sporadic task, the distance between the arrival times of two consecutive jobs is greater than or equal to T_i, $\forall k, a_{i.k+1} - a_{i,k} = T_i$. Moreover, the deadline is computed as $d_{i,k} = a_{i,k} + D_i$, and the computation time never exceeds the worst case computation time, $\forall k, c_{i,k} \leq C_i$. In this paper we restrict our attention to sporadic tasks with constrained or implicit deadline, i.e., having $D_i \leq T_i$, or $D_i = T_i$, respectively.

The *hyperperiod* is computed as the least common multiple of the task periods: $H = \mathsf{lcm}(T_1, T_2, \ldots, T_n)$.The utilisation of a task τ_i is defined as $U_i = C_i/T_i$. The total utilisation of a task set τ composed of n sporadic tasks is denoted as $U = \sum_{i=0}^{n} U_i$.

The demand bound function $\mathsf{dbf}(t)$ is defined as the total amount of computation time of the tasks that have arrival time and deadline in $[0, t]$. For a set of real-time sporadic tasks, the dbf function can be computed as

$$\mathsf{dbf}(t) = \sum_{i=1}^{n} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i. \tag{1}$$

Notice that the dbf is a step-wise function which changes values at the absolute deadlines of the jobs. We define as $\mathsf{dSet}(L)$ the set of all absolute deadlines of

all jobs in interval $[0, L]$:

$$\mathsf{dSet}(L) = \{d_{i,k} | d_{i,k} \leq L\}. \tag{2}$$

The following theorem gives a necessary and sufficient condition for schedulability.

Theorem 1. *A set of sporadic real-time tasks $\mathcal{T}$ is schedulable on a single processor by the earliest deadline first scheduling algorithm if and only if:*

$$\forall t \in dSet(H) \quad \mathsf{dbf}(t) \leq t \tag{3}$$

For a constrained deadline task set with $U < 1$, when the computation times of all tasks are known, it is possible to reduce the amount of deadlines to be checked to the first busy period, or to interval $[0, L^*]$, as shown in [1], where

$$L^* = \frac{\sum_{i=1}^{n} U_i(T_i - D_i)}{1 - \sum_{i=1}^{n} U_i}.$$

3 Problem statement

Suppose that we have a task set where we know the periods and the relative deadlines, whereas the worst-case computation times are unknown. Our goal is to compute the *C-space* [3], i.e., all the possible values of the worst-case computation times that make the system schedulable.

To do this, we set-up a system of inequalities using Theorem 1. In particular, we define

$$n_i(t) = \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1,$$

i.e. the number of instances of task τ_i entirely contained in interval $[0, t]$. Then the inequality can be written as:

$$\begin{cases} \forall t \in \mathsf{dSet}(\mathcal{T}) & \sum_{i=1}^{n} n_i(t) C_i \leq t \\ \forall i = 1, \ldots, n, & C_i \geq 0 \end{cases} \tag{4}$$

This set of inequality defines a convex polyhedron in the space of the variables C_i: all solutions to this set of inequalities are the vectors that we are looking for.

Since computation times are unknown, we have to analyse all deadlines in $\mathsf{dSet}(H)$. However, the number of points in $\mathsf{dSet}(H)$ may be very large, due to a large hyperperiod. In turns, every point in $\mathsf{dSet}(H)$ corresponds to a linear inequality. Therefore, it is natural to ask if all such inequalities are strictly necessary.

An inequality can be safely removed if, by doing so, no new solution is introduced. George et al. [3] have shown that indeed it is possible to eliminate many such inequalities. In their paper, they use the Simplex algorithm, to identify the inequalities to remove. Their approach is presented in section 4.

Another equivalent method is to use a geometric characterisation of the polyhedron and computing the smallest convex enveloping polyhedron. We will describe this method in more details in the Section 6.

How many inequalities are removed by applying the elimination procedure? How many are left? Is there any special pattern to identify the minimum set of inequalities, to help us reduce the complexity of the problem *in the average case*?

This paper reports some early investigation on this problem. While we have no definitive answer to these questions, we believe that writing the known facts that we discovered by performing extensive experiments may convince other researchers to help us solve the problem.

4 Linear Programming approach

In [3], George et al. show how to characterise the space of feasible WCETs. Considering the WCETs in $X = (x_1, \ldots, x_n)$ as variables and D, T constants, the C-space is defined by a set of $s+1$ constraints, where the first s constraints are derived from the inequalities in Equation 4 corresponding to absolute deadlines in $\mathsf{dSet}(H)$ and the $(s+1)^{th}$ constraint is derived from the load utilisation ($U \leq 1$).

They show how to prune the set $\mathsf{dSet}(H)$ to extract the subset of absolute deadlines representing the most constrained times characterising the C-space. For any time $t_i \in \mathsf{dSet}(H)$, they formalise as a linear programming problem in which the constraint corresponding to t_i is removed, and the objective function is to maximise $dbf(t_i)$. More formally:

Linear Programming Problem: LP 1

Maximise $dbf(t_i)$
With $x_1 \geq 0, \ldots, x_n \geq 0$ positive real variables
Under the constraints:
$\bigcap_{k=1,k\neq i}^{s} \{dbf(t_k) \leq t_k\}$

In [3], the author show that the space of feasible WCETs is convex. Hence, LP1 can be solved by using the Simplex Algorithm. If the solution of the problem is $dbf(t_i) \leq t_i$, then the corresponding inequality (that was just removed to obtain LP1) is redundant, and can safely be eliminated. In fact, the maximum value that the $dbf(t_i)$ can assume is however inferior to t_i even without imposing the constraint. On the other hand, if $dbf(t_i) > t_i$, the corresponding constraint cannot be eliminated without enlarging the C-space.

4.1 Numerical Example

We consider a sporadic task set $\tau = \{\tau_1, \tau_2, \tau_3\}$, composed of three sporadic tasks τ_i, where, for any task τ_i, T_i and D_i are fixed, and $x_i \in \mathbb{R}^+$, the WCET of task τ_i, is variable.

– τ_1 : $(x_1, T_1, D_1) = (x_1, 7, 5)$;
– τ_2 : $(x_2, T_2, D_2) = (x_2, 11, 7)$;
– τ_3 : $(x_3, T_3, D_3) = (x_3, 13, 10)$.

In this example, we have: $D_{min} = 5$ and $H = 1001$. To characterise the C-space, we have to consider all absolute deadlines in $\mathsf{dSet}(H)$ in time interval $[5, 1001)$. The cardinality m of $\mathsf{dSet}(H)$ is 281.

Therefore, we apply the simplex algorithm on the Linear Programming problem LP_i, for any time $t_i \in \mathsf{dSet}(H)$, starting from time t_m down to time t_1 (to optimise the computation). We obtain the following subset $\mathcal{S}_1$ of times in $\mathsf{dSet}(H)$:

$$\mathcal{S}_1 = \{5, 7, 10, 12, 19, 40\} \subseteq \mathcal{M}.$$

Since:

$$\begin{cases} x_1 + x_2 \leq 7 \\ 2\,x_1 + x_2 + x_3 \leq 12 \end{cases} \Rightarrow\ 3\,x_1 + 2\,x_2 + x_3 \leq 19$$

the set can be further reduced to:

$$\mathcal{S}_2 = \{5, 7, 10, 12, 40\} \subseteq \mathcal{S}_1.$$

The exact deadline set characterising the C-space is therefore $\{5, 7, 10, 12, 40\}$.

4.2 Experiments

We now study the performance of the simplex for pruning the elements in S, in the case of constrained deadlines ($\forall i \in [1, n], D_i \leq T_i$).

The first question we ask is how large is the number of points in dSet before the reduction. This number strongly depends on the hyperperiod: if periods are co-prime, the hyperperiod can be very large. Suppose that all periods are prime with respect to each other. Then the total number of deadline points generated by task i is $\prod_{j=1, j\neq i}^{n} T_j$, and the total number of points is:

$$|\mathsf{dSet}| = \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} T_j.$$

Of course, if the periods are multiples of each other, the hyperperiod and the number of points become much lower.

Notice that the number of constraints in $\mathsf{dSet}(H)$ depends more on the value of the periods, than on the number of tasks (the number of constraints can be small even for a high number of tasks).

We generated 100,000 task systems, each one containing 3 tasks. For each system, we proceed as follows:

– The period of each task is uniformly chosen from $[1, 100]$

– The deadline of any task $\tau_i(C_i, T_i, D_i)$ is $D_i = \alpha T_i$. α is chosen in the intervals $[0, 0.8]$ and $[0.8, 1]$ with a granularity of respectively 0.1 and 0.025.

We focus on the influence of α on the size of the $\mathsf{dSet}(H)$ after pruning the constraints.

Figure 1 shows the results of our analysis. The average number of elements in $\mathcal{S}$ over all generated systems is represented by the solid line and associated to the left axis as a function of α. The dotted line refers to the average number of elements obtained after the simplex is applied to the linear programming problem LP1 and must be read according to the right axis, as a function of α.

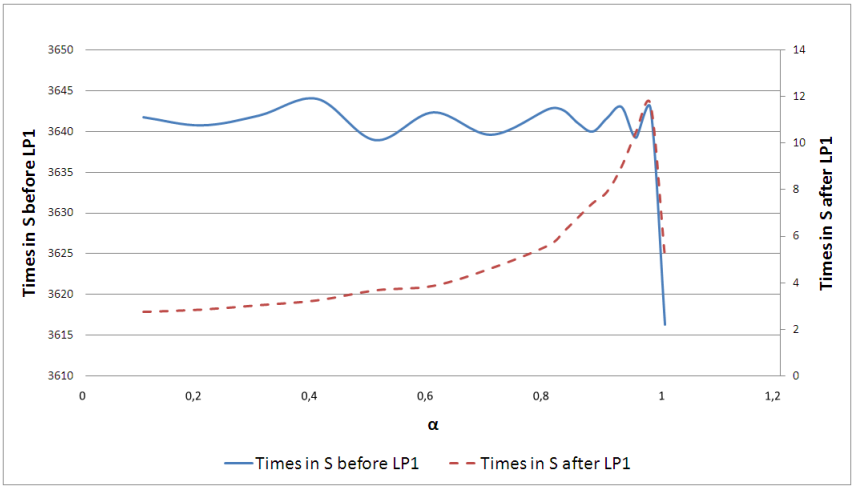

Fig. 1. Reduction of elements in $\mathcal{S}$ with LP1

We notice that the number of elements which curb the C-Space inch-up in $[0.1, 0.6]$ then descend when α tends toward 1. If $\alpha = 1$, we are in the special case of implicit deadlines where the only constraint is the processor utilisation constraint: $U = \sum_{i=1}^{n} \frac{x_i}{T_i} \leq 1$.

In all generated systems with $\alpha < 1$, we have found that the number of constraints before and after pruning the set $\mathcal{S}$ is respectively higher than 3570 and less than 12. For a load less then 0.6, the average number of constraints in $\mathcal{S}$ after pruning the elements in $\mathcal{S}$ is less than or equal to 4. Similar trends are confirmed for larger task sets.

This confirms that the average complexity of the feasibility problem is much smaller than the worst-case. The Simplex is very effective in this reduction. However, the Simplex is itself an algorithm with high computational complexity in the worst-case: and the Simplex must be run for every point in $\mathsf{dSet}(\mathsf{H})$. Therefore, even if the final number of points is very low, to obtain such a reduction we need to apply a complex algorithm an exponential number of times.

We then need to investigate if there is some other property of the problem that can be used to reduce the number of points.

5 The definitive idle time approach

One explanation for the increasing trend in Figure 1 can be given by the presence of definitely idle times.

Definition 1. *A time t_d is said to be a **definitive idle time** (DIT) if at time t_d there is no task released before t_d having an absolute deadline after t_d.*

A DIT must remain idle for any value of the computation times that makes the task set schedulable. In facts, a DIT is a time in which no schedulable job can execute.

A DIT exists only if all tasks in the task set have constrained or implicit deadlines (i.e. $\forall i, D_i \leq T_i$). In this case, one DIT is point H, because at the hyperperiod all jobs must have terminated, and no new job has arrived yet.

Consider, as an example, the task set consisting of two tasks, $\tau_1 = (D_1 = 5, T_1 = 8)$ and $\tau_2 = (D_2 = 9, T_2 = 15)$. All times in $[13, 15]$ are definitely idle because τ_1 has jobs $J_{11} = (0, 5)$, $J_{12} = (8, 13)$, $J_{13} = (16, 21)$, ..., and τ_2 has jobs $J_{21} = (0, 9)$, $J_{22} = (15, 24)$, ..., and there is no job active in interval $[13, 15]$. Therefore, for any value of C_1 and C_2 for which the task set is schedulable, interval $[13, 15]$ will remain idle, and hence any initial busy period has length less than 13.

It is easy to show that all constraints corresponding to deadlines after $t = 13$ are redundant.

Lemma 1. *Let t_1 be a definitely idle time. Then for any $t > t_1$, $\mathsf{dbf}(t) \leq t$ is a redundant inequality.*

Proof. Since t_1 is a DIT, for any $t > t_1$ we can rewrite:

$$\mathsf{dbf}(t) = \mathsf{dbf}(t_1) + \mathsf{dbf}(t - t_1) = \mathsf{dbf}(d_1) + \mathsf{dbf}(d_2)$$

where d_1 is the latest deadline no later than t_1, and d_2 is the latest deadline no later than $(t - t_1)$ after a critical instant, i.e., the synchronous periodic arrival of all task instances.

Then

$$\begin{cases} \mathsf{dbf}(d_1) \leq d_1 \\ \mathsf{dbf}(d_2) \leq d_2 \end{cases} \Rightarrow \quad \mathsf{dbf}(t) \leq d_1 + d_2 \leq t$$

□

Clearly, the smaller the ratio between relative deadlines and periods, the highest is the probability that the first DIT happens quite soon in the schedule. Hence, one possible reason for the trend in Figure 1.

Consider again the previous example: time 13 is the first DIT, hence only deadlines $\mathsf{dSet}(13) = \{5, 9, 13\}$ have to be considered, and in fact the corresponding three inequalities are all non redundant. Therefore, thanks to the concept of DIT, in this particular example we defined precisely the minimum set of inequalities. However, this is not true in general. Consider the example task set in Table 1.

$Tasks$	D	T	Jobs Windows
τ_1	7	9	$[0, 7], [9, 16], [18, 25], [27, 34]$
τ_2	12	15	$[0, 12], [15, 27], [30, 42]$

Table 1. Example 2

The first DIT is at time 27, and all deadlines in $[0, 27]$ are $\mathsf{dSet}(27) = \{7, 12, 16, 25, 27\}$: of these, the inequalities corresponding to $\{7, 12, 16, 27\}$ are necessary, while the inequality corresponding to deadline 25 is redundant. In fact,

$$\begin{cases} 2C_1 + C_2 \leq 16 \\ C_1 \leq 9 \end{cases} \quad \Rightarrow \quad 3C_1 + C_2 \leq 25.$$

Therefore, not all deadlines before the first DIT correspond to necessary inequalities. As the ratio between deadlines and periods approaches 1, and as the number of tasks n increases, the first DIT happens quite late in the schedule; nevertheless, many inequalities are still redundant, and the minimum number of necessary inequalities is small.

5.1 First definitive idle time computation

Notice that $t_d = H$ is a DIT in the constrained deadlines case. Indeed, $W(H) = U \cdot H$ and

$$n_i(H) = \left\lfloor \frac{H + T_i - D_i}{T_i} \right\rfloor = \frac{H}{T_i} + \left\lfloor \frac{T_i - D_i}{T_i} \right\rfloor = \frac{H}{T_i}$$

as $D_i \leq T_i$. Thus $dbf(H) = U \cdot H = W(H)$: for constrained deadlines, H is not necessarily the first DIT.

Our goal now is to propose an algorithm to compute the first DIT before H, if it exists. For a task τ_i, a first DIT t_d corresponds to a time between a deadline of τ_i and the next release time of τ_i after this deadline. This corresponds modulo the task period to a time between $[D_i, T_i]$.

Thus, finding the first DIT is equivalent to find the first time t_d satisfying:

$$\begin{aligned} \forall i, t_d &= a_i \; Mod \; T_i \\ a_i &\in [D_i, T_i] \end{aligned} \tag{5}$$

This problem can be solved by using the general Chinese Remainder Theorem.

Theorem 2 (General Chinese Remainder Theorem). *Suppose $T_1, T_2, \ldots, T_n$ are positive integers which are pairwise coprime. Then, for any given sequence of integers $a_1, a_2, \ldots, a_n$, there exists an integer x solving the following system of simultaneous congruences.*

$$\begin{aligned} x &\equiv a_1 \pmod{T_1} \\ x &\equiv a_2 \pmod{T_2} \\ &\vdots \\ x &\equiv a_n \pmod{T_n} \end{aligned}$$

Furthermore, all solutions x of this system are congruent modulo the product, $H = T1T2 \ldots T_n$

Notice that the theorem requires all integers to be pairwise coprime. In this particular case, one DIT is the solution of:

$$t_{idle} = \sum_{i=1}^{n} a_i \frac{H}{T_i} \left[\left(\frac{H}{T_i} \right) \right]_{T_i}^{-1} \pmod{T_1 T_2 \cdots T_n} \quad for\ a_i \in [D_i, T_i] \tag{6}$$

where $[a^{-1}]_b$ stands for the multiplicative inverse of $a \pmod b$ i.e. the number y such that $ay = 1 \pmod b$. t_d is thus the minimum value satisfying equation 5 i.e:

$$t_d = min_{a_i \in [D_i, T_i]} t_{idle} \tag{7}$$

If we compute the first definitive idle time in the example used for the linear programming approach where periods are all pairwise coprimes and $H = T_1 T_2 T_3 = 1001$:

- τ_1 : $(x_1, T_1, D_1) = (x_1, 7, 5)$;
- τ_2 : $(x_2, T_2, D_2) = (x_2, 11, 7)$;
- τ_3 : $(x_3, T_3, D_3) = (x_3, 13, 10)$.

We find the minimum value of t_{idle} for $a_1 = 6, a_2 = 7$ and $a_3 = 10$:

$$\begin{aligned} t_d &= a_1 T_2 T_3 [(T_2 T_3)^{-1}]_{T_1} + a_2 T_1 T_3 [(T_1 T_3)^{-1}]_{T_2} + a_3 T_1 T_2 [(T_1 T_2)^{-1}]_{T_3} \\ &= 6 \cdot 11 \cdot 13 \cdot 5 + 7 \cdot 7 \cdot 13 \cdot 4 + 10 \cdot 7 \cdot 11 \cdot 12 = 16078 \end{aligned}$$

and $16078 \pmod{T_1 T_2 T_3} = 62$. Thus $t_d = 62$. Hence, according to this approach, we only need to consider the deadlines in $[0, 62]$. To these, we can apply the Simplex algorithm, thus reducing considerably the time to derive the minimum set of inequalities.

In the general case where periods are not coprime, it is possible to use one of the algorithms for solving systems of simultaneous congruences, for example the method of successive substitutions.

Thus, applying the definitive idle time approach can help reducing the initial time interval to consider before applying the linear programming approach.

6 Geometric interpretation

We now look at the problem from a slightly different point of view. As discussed in Section 3, for each deadline $d \in \mathsf{dSet}(L)$, we have an inequality of the form

$$\sum_{i=1}^{n} n_i(d) C_i \leq d$$

where $n_i(d)$ is the number of jobs of τ_i entirely included in interval $[0, d]$. Using some simple algebra, we can rewrite the inequality as:

$$\sum_{i=1}^{n} \frac{n_i(d) T_i}{d} U_i \leq 1$$

In vectorial form:

$$\mathbf{V}(d) \cdot \mathbf{U} \leq 1 \tag{8}$$

where $\mathbf{V}(d)$ is the vector whose i-th component is $\frac{n_i(d)T_i}{d}$, $\mathbf{U}$ is the vector of task utilisations, and the dot represents scalar product. Hence, the set of deadlines defines a set of vectors $\mathbf{V}$ in the space of the task utilisations.

A vector is redundant if it can be expressed as a linear combination of other vectors with positive coefficient whose sum is less than 1.

Theorem 3. *A vector $\mathbf{V}(d)$ is redundant is there exists at least two other vectors $\{\mathbf{V}(d_1), \ldots, \mathbf{V}(d_h)\}, h \geq 2$ such that*

$$\mathbf{V}(d) = k_1 \mathbf{V}(d_1) + \ldots + k_k \mathbf{V}(d_h), \quad \sum_{j=1}^{h} k_j \leq 1$$

Proof. Equation (8) must be true for all vectors. We show that under the hypothesis, the inequality for d can be eliminated.

$$\mathbf{V}(d) \cdot \mathbf{U} = \sum_{j=1}^{h} k_j \mathbf{V}(d_j) \cdot \mathbf{U} \leq \sum_{h=1}^{j} k_j \leq 1$$

□

The implication of the Theorem is that the set of necessary points deadlines is identified by the convex hull of all the points $\mathbf{V}(d)$, i.e. the most external points that define a convex polyhedron in the utilisation space. To clarify the issue, let us make an example.

Consider the task set of Table 2. The first DIT is at 38, hence the set of deadlines to consider is: $\mathsf{dSet}(38) = \{6, 12, 14, 22, 25, 30, 38\}$. Each of these deadlines corresponds to a vector in the space of utilisations $< U_1, U_2 >$. The resulting space is shown in Figure 2.

As a result, the minimal set of deadlines to consider is $\mathsf{dSet}^* = \{6, 12, 14, 38\}$. Unfortunately, computing the convex hull is in general rather complex. Specialised algorithms exist in 2 and 3 dimensions that have complexity $O(s \log(s))$

Tasks	D	T	Jobs
τ_1	6	8	$[0, 6], [8, 14], [16, 22], [24, 30], [32, 38]$
τ_2	12	13	$[0, 12], [13, 25], [26, 38], [39, 51]$

Table 2. Example 3

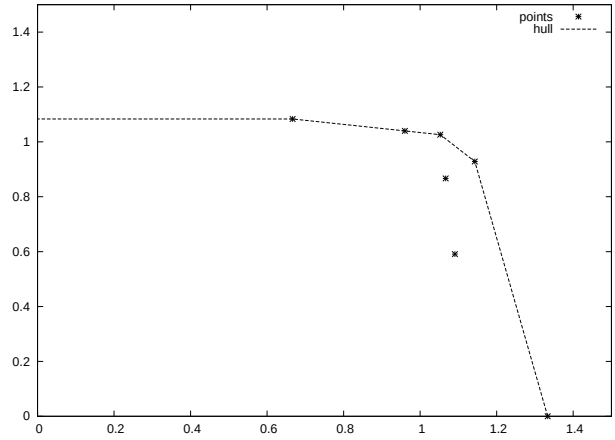

Fig. 2. Representation of $\mathbf{V}(d)$ in the utilisation space for the tasks in Table 2, together with the convex hull.

and $O(s^2)$, respectively, where s is the number of points in **dSet**. However, in the n-dimension case the complexity is $O(s^{\lfloor \frac{n}{2} \rfloor + 1})$, therefore it grows exponentially in the number of constraints. The DIT is even more important in the case, as it can help reduce s before applying the convex-hull algorithm.

7 Conclusion

In this paper we have described the problem of computing the C-Space of a set of sporadic tasks scheduled by EDF on a single processor. The method amounts to using the Processor Demand Analysis for generating a set of inequalities that define the space. We have seen that in practical situations, many of the inequalities are redundant and can hence be eliminated, reducing the complexity of the C-Space representation. However, methods for reducing the number of inequalities are themselves complex and require a certain amount of computation.

We have observed that the concept of Definitely Idle Time can help us reducing the number of constraints to analyse in the case of constrained deadline systems. The method is particularly effective when the differences between the relative deadlines and the periods are large. Then, we have proposed a different point of view on the problem by using a geometric interpretation.

The observations reported in this paper are important for better understanding the complexity of feasibility analysis for single processor systems. While the

problem has been proved to be NP-hard, the facts reported in this paper show that in practical cases the complexity is rather low. These facts are in accordance with other results on similar problems, as reported by Zhang and Burns [4].

We believe, however, that the investigation is not concluded. In addition to the Definitely Idle Time, other properties could be used to further reduce the set of non-redundant deadlines quickly and more effectively. The results could shed light on more fundamental properties of the feasibility problem.

References

1. Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. pages 182–190, 1993.
2. S.K. Baruah, L.E. Rosier, and R.R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *The Journal of Real-Time Systems*, 2, 1990.
3. L. George and J.F. Hermant. Characterization of the space of feasible worst-case execution times for earliest-deadline-first scheduling. *Journal of Aerospace Computing, Information, and Communication*, 6(11):604–623, 2009.
4. F. Zhang and A. Burns. Schedulability analysis for real-time systems with edf scheduling. *Computers, IEEE Transactions on*, 58(9):1250–1258, 2009.