Improvements in Polynomial-Time Feasibility Testing for EDF

Alejandro Masrur  Sebastian Drössler  Georg Färber
Institute for Real-Time Computer Systems
Technische Universität München, Germany
{Alejandro.Masrur, Sebastian.Droessler, Georg.Faerber}@rcs.ei.tum.de

Abstract

This paper presents two fully polynomial-time sufficient feasibility tests for EDF when considering periodic tasks with arbitrary deadlines and preemptive scheduling on uniprocessors. Both proposed methods are proven, analytically and by means of an extensive experimental comparison, to be more accurate than known polynomial-time feasibility tests. Additionally, we show for a wide interval of practical processor utilization that one of these methods presents almost the same efficiency, in terms of accepted task sets, as the more complex pseudo-polynomial-time exact feasibility tests.

1. Introduction

For the case where periodic tasks with arbitrary deadlines are preemptively scheduled under EDF (Earliest Deadline First) on a uniprocessor, exact feasibility tests with pseudo-polynomial complexity are already known, e.g. [2] and [8]. However, faster polynomial-time sufficient tests are still preferable for some applications where, for example, schedulability has to be tested online. Furthermore, the more predictable running time of polynomial-time algorithms is an additional advantage in the latter context.

In this paper, we show how it is possible to achieve more accuracy in polynomial-time feasibility testing for EDF under the given conditions. We propose two feasibility tests basing actually on same idea as the density condition [10], [7] and as Devi’s condition [4], i.e., to reduce algorithms’ complexity by sacrificing exactness.

The next two sections provide a survey of related work and a description of the used task model. Posteriorly, the tight relation between Devi’s condition and the best known feasibility bound is pointed out in section 4. In section 5, we prove analytically that a slight, but for the purpose of this paper profitable, improvement of the best known feasibility bound is possible. Section 6 presents two better fully polynomial-time feasibility tests for EDF; one of them exhibits complexity $O(n^2)$ while the other has same complexity as Devi’s condition, namely $O(n \log n)$. Additionally, section 7 outlines results from thorough experiments comparing our proposed methods against the best known feasibility tests for EDF. Finally, some concluding remarks are presented in section 8.

2. Related work

In [6], Liu and Layland showed that a feasibility test for synchronous tasks, scheduled preemptively under EDF and on a uniprocessor, can be performed as follows in polynomial time, when deadline $(d_i)$ is equal to period $(p_i)$ for every task: $U = \sum_{i=1}^{n} \frac{d_i}{p_i} \leq 1$. In this inequality, $e_i$ is the worst-case execution time of the $i$-th task, $n$ is the number of tasks and $U$ is called processor utilization. Afterwards in [2], Baruah et al. proved that applying the utilization test of Liu and Layland is also valid when $d_i \geq p_i$ holds for all $i$.

When deadlines are allowed to be less than periods, the complexity grows considerably. However, assuming the processor utilization to be less than 100% for a synchronous scheduling, Baruah et al. also proved that if a deadline is missed, this happens before a maximum time upper limit known as feasibility bound. This result allowed Baruah et al. to design a pseudo-polynomial time algorithm for the case where deadlines are not forced to be equal to periods.

Another pseudo-polynomial time algorithm for $d_i \leq p_i$ was presented by Ripoll et al. in [8]. Ripoll et al. introduced two better feasibility bounds, which they combined in an efficient algorithm. On the other hand, George et al. considered in [5] also the case $d_i > p_i$ and got a feasibility bound, referred as George’s bound in this paper, with the expression: $I_n = \sum_{i=1}^{n} \frac{(p_i - \min(d_i, p_i)) \cdot u_i}{1 - U}$, where $u_i = \frac{d_i}{p_i}$. George’s bound reduces to Ripoll’s bound if $d_i \leq p_i$ holds for all possible $i$ and it is the best known feasibility bound for EDF in case that no restrictions are imposed to $d_i$.

The other feasibility bound presented by Ripoll et al. is based on the busy period analysis, whose calculation itself presents pseudo-polynomial complexity. This lat-
ter pseudo-polynomial feasibility bound was also independently obtained by Spuri [9].

All exact algorithms, including the one of Albers and Slomka [1], present pseudo-polynomial complexity. In order to reach polynomial complexity in feasibility testing for EDF, when \( d_i \) can be less than \( p_i \), exactness must be sacrificed. Based on this idea, Liu in [7] and Stankovic et al. in [10] propose independently the density test, which results by replacing \( \frac{1}{p_i} \) with \( \min(d_i, p_i) \) in the utilization test of Liu and Layland, that is: \( \sum_{i=1}^{n} \frac{d_i}{\min(p_i, d_i)} \leq 1 \).

Assuming that tasks are sorted by non-decreasing deadlines \( d_i \), Devi presented in [4] a better approach in terms of accepted task sets than the density condition. However, Devi’s condition describes a higher complexity \( O(n \log n) \) because of requiring sorted task sets.

3. Task model and notation

In this section, we specify some more details about the task model and notation already used for related work.

We consider a set \( \tau \) of periodic real-time tasks, which are fully preemptable and independent. As already mentioned, we assume that \( \tau \) is scheduled on a uniprocessor. Each task \( T_i \) in \( \tau \) is characterized by its period of repetition \( p_i \), its relative deadline \( d_i \), and its worst-case execution time \( c_i \). Another parameter of tasks is the initial release time or phase \( \phi_i \). For this paper, it is assumed that \( \phi_i = 0 \) holds for all tasks, i.e. \( \tau \) is a synchronous task set. Relative deadlines \( d_i \) are not restricted and they can be less as well as greater than the respective periods \( p_i \) for \( 1 \leq i \leq n \), where \( n \) is the number of tasks in \( \tau \).

For remaining sections, we assume that the total processor utilization given by \( \tau \) is less than 100\%, i.e. \( U < 1 \) holds. In addition, the time is represented by the real-valued \( t \geq 0 \), whereas George’s bound for \( \tau \) is denoted by \( I_n \).

4. Devi’s condition and George’s bound

The relation between Devi’s algorithm and the best known feasibility bound was indirectly shown in [1]. However, the following lemma formulates this relation in a more explicit way.

**Lemma 1** Assuming that tasks in \( \tau \) are sorted according to non-decreasing relative deadlines, Devi’s condition is equivalent to an iterative calculation of George’s feasibility bound. The task set \( \tau \) is feasible, if George’s bound for the first \( k \) tasks is less than or equal to \( d_k \) for every possible \( k \), where \( 1 \leq k \leq n \).

**Proof:** It is assumed that tasks are sorted according to non-decreasing relative deadlines, so if \( i < j \) holds, \( d_i \leq d_j \) will also hold. Devi’s condition states that \( \tau \) is feasible if the following inequality holds for every possible \( k \), where \( 1 \leq k \leq n \) [4]:

\[
\sum_{i=1}^{k} \frac{d_i}{p_i} + \frac{1}{d_k} \sum_{i=1}^{k} \left( \frac{p_i - \min(p_i, d_i)}{p_i} \right) \cdot c_i \leq 1.
\]

Reordering terms, we get:

\[
\sum_{i=1}^{k} \left( \frac{p_i - \min(p_i, d_i)}{p_i} \right) \cdot c_i \leq \left( 1 - \sum_{i=1}^{k} \frac{d_i}{p_i} \right) \cdot d_k,
\]

and finally:

\[
\sum_{i=1}^{k} \left( \frac{p_i - \min(p_i, d_i)}{p_i} \right) u_i \leq \frac{d_k}{1-U_k} \cdot d_k.
\]

Where \( U_k = \sum_{i=1}^{k} \frac{d_i}{p_i} \) and \( u_i = \frac{c_i}{p_i} \), i.e. the left side of this latter inequality is George’s bound \( I_k \) for the first \( k \) tasks. The thesis follows.

Lemma 1 suggests that improving the best known feasibility bound also results in better polynomial-time feasibility tests. This aspect is analyzed in the next sections.

5. Improving George’s bound

In this section, we prove that it is possible to reach a slight but anyway useful improvement of the best known feasibility bound. For the sake of clarity, the following lemma restates one of the theorems presented by George et al. in [5].

**Lemma 2** If the schedule of a given synchronous task set \( \tau \) is not feasible, i.e. a deadline is missed for the first time at \( t_{\text{miss}} \), then \( t_{\text{miss}} < I_n \), holds, where \( I_n \) is George’s bound.

**Proof:** This lemma simply excludes the possibility that \( t_{\text{miss}} = I_n \). Because a deadline is missed at \( t_{\text{miss}} \), the total execution demand of \( \tau \) at \( t_{\text{miss}} \) is greater than the available time \( t_{\text{miss}} \), i.e. \( t_{\text{miss}} < h(t_{\text{miss}}) \). Where \( h(t) \) is the demand bound function defined by Baruah et al. in [2] and has the expression:

\[
h(t) = \sum_{i=1}^{n} \max \left( 0, \left[ \frac{t-x}{p_i} + 1 \right] \right) \cdot c_i.
\]

Considering \( \min(p_i, d_i) \) instead of \( d_i \), we obtain:

\[
t_{\text{miss}} < \sum_{i=1}^{n} \left( \frac{t_{\text{miss}} - \min(p_i, d_i)}{p_i} \right) + 1 \right) \cdot c_i.
\]

Removing the floor function, we reach:

\[
t_{\text{miss}} < \sum_{i=1}^{n} \left( \frac{t_{\text{miss}} - \min(p_i, d_i)}{p_i} + 1 \right) \cdot c_i.
\]

Reshaping this inequality to obtain \( t_{\text{miss}} \), we get the expression:

\[
t_{\text{miss}} < I_n.
\]

In words, George’s bound is always greater than the time instant at which a deadline is missed for the first time.

**Lemma 3** If a deadline is missed for the first time at \( t_{\text{miss}} \), then \( t_{\text{miss}} < I_n \), where \( u_i = \frac{c_i}{p_i} \), \( U_x = \sum_{i=1; i \neq x}^{n} u_i \) and \( c_x(0) \) is given by

\[
max \left( 0, \left[ \frac{t-x}{p_x} \right] \right).
\]

**Proof:** We know from lemma 2 that \( t_{\text{miss}} \) is always less than \( I_n \). Consequently, \( c_x(0) = max \left( 0, \left[ \frac{t-x}{p_x} \right] \right) \).
is equal to or greater than the amount of jobs of task $T_x$ in the interval $[0, t_{miss}]$. Additionally, as a deadline is missed at time $t_{miss}$, the total execution demand in $[0, t_{miss}]$ is greater than $t_{miss}$, i.e. $t_{miss} < \sum_{i=1; i \neq x}^{n} \max \left(0, \left\lfloor \frac{t_{miss} - d_i}{p_i} \right\rfloor + 1 \right) \cdot e_i + c_x(0) \cdot e_x$. By removing the floor function and reshaping to get $t_{miss}$, we obtain:

$t_{miss} < \frac{c_x(0) \cdot e_x + \sum_{i=1; i \neq x}^{n} (p_i \cdot \min(p_i, d_i)) \cdot u_i}{1 - U_{-1; \ldots; -k}}$. \hfill \Box$

**Lemma 4** $I_n > t_{n}^{(1)} = c_x(0) \cdot e_x + \sum_{i=1; i \neq x}^{n} (p_i \cdot \min(p_i, d_i)) \cdot u_i$ holds, where $u_i = \frac{e_i}{p_i}, U_{-x} = \sum_{i=1; i \neq x}^{n} u_i$, and $c_x(0)$ is given by max $\left(0, \left\lfloor \frac{t_{miss} - d_x}{p_x} \right\rfloor \right)$. $I_n$ is equal to $I_n(0)$ holds. The thesis follows. \hfill \Box

**Proof:** From the derivation of $I_n$, we know that the approximated total execution demand, which results by removing the floor function and considering $\min(p_i, d_i)$ instead of $d_i$, is equal to $I_n$. This can be mathematically expressed as follows:

$I_n = \sum_{i=1; i \neq x}^{n} \left( \frac{t_{miss} - \min(p_i, d_i)}{p_i} + 1 \right) \cdot e_i + c_x' \cdot e_x$.

In the previous inequality, we have substituted the term $t_{miss} - \min(p_i, d_i)$ by $c_x'$ and reshaping to get $I_n$, we have:

$I_n = \frac{c_x' \cdot e_x + \sum_{i=1; i \neq x}^{n} (p_i \cdot \min(p_i, d_i)) \cdot u_i}{1 - U_{-x}}$.

Finally, it is easy to see that $c_x' > c_x(0)$ holds, where $c_x(0)$ is equal to max $\left(0, \left\lfloor \frac{t_{miss} - d_x}{p_x} \right\rfloor \right)$. The thesis follows. \hfill \Box

**Theorem 1** If a deadline is missed for the first time at $t_{miss}$, then $t_{miss} < I_n(k)$ holds. Conversely, considering $x = 1$ in Lemma 3, we have that $c_1(0) = \max \left(0, \left\lfloor \frac{t_{miss} - d_1}{p_1} \right\rfloor \right)$ is equal to or greater than the amount of jobs of task $T_1$ in the interval $[0, t_{miss}]$. In the same way, $c_2(1) = \max \left(0, \left\lfloor \frac{t_{miss} - d_2}{p_2} \right\rfloor \right)$ is equal to or greater than the amount of jobs of task $T_2$ in $[0, t_{miss}]$. We proceed analogously to Lemma 3 to get $t_{miss} < I_n(2)$ holds. Proceeding recursively for the first $k$ tasks and $1 \leq k \leq n$, we get:

$t_{miss} < \sum_{i=1; i \neq x}^{n} c_i^{(i-1)} \cdot e_i$ and $t_{miss} < I_n(n) = \sum_{i=1}^{n} c_i^{(i-1)} \cdot e_i$. \hfill \Box

**6. Polynomial-time feasibility tests for EDF**

In this section, we prove that it is possible to build polynomial-time sufficient feasibility tests for EDF basing on theorem 1.

**Lemma 5** Assuming that tasks in $\tau$ are sorted according to non-decreasing relative deadlines, $\tau$ is feasible under EDF if $I_k^{(1)} \leq d_k$ holds for every $k$, where $1 \leq k \leq n$ and $I_k^{(1)}$ is the feasibility bound of theorem 1 for the first $k$ tasks.

**Proof:** Let us analyze the case $k = 1$—only for $T_1$; if $I_1^{(1)} \leq d_1$ holds, there are no deadlines in $[0, I_1^{(1)})$, because tasks are sorted according to non-decreasing relative deadlines—$d_1$ is the shortest one, and theorem 1 excludes the possibility that a deadline miss happens at $t = I_1^{(1)}$. As a consequence, the task set with only $T_1$ is feasible. For $k = 2$, we compute $I_2^{(2)}$—only for $T_1$ and $T_2$; if $I_2^{(2)} \leq d_2$ holds, the task set with only $T_1$ and $T_2$ is also feasible because $T_1$ alone was proven to be feasible in the previous step and there are no deadlines of $T_2$ in $[0, I_2^{(2)})$. Moreover, a deadline miss cannot happen at $t = I_2^{(2)}$ according to theorem 1. For $k = 3$, we compute $I_3^{(3)}$—only for $T_1, T_2$ and $T_3$; if $I_3^{(3)} \leq d_3$ holds, the first three tasks are also feasible because $T_1$ and $T_2$ together were proven to be feasible in the previous step and there are no deadlines of $T_3$ in $[0, I_3^{(3)})$. A deadline miss cannot happen at $t = I_3^{(3)}$ because of theorem 1.

Now, let us assume that $I_k^{(k)} \leq d_k$ holds for $1 \leq k \leq n - 1$. If $I_n^{(n)} \leq d_n$ also holds, $\tau$ is feasible because the first $n - 1$ tasks were proven to be feasible in the previous steps and there are no deadlines of $T_n$ in $[0, I_n^{(n)})$. In accordance to theorem 1, a deadline miss can neither happen exactly at $t = I_n^{(n)}$. The thesis follows. \hfill \Box

**Lemma 6** A polynomial-time feasibility test based on theorem 1 is more accurate than Devi’s condition.
6.1. Feasibility test with $O(n^2)$

Figure 1 presents an algorithm with $O(n^2)$ denominated $ptftn^2$, which bases on lemma 5. Theorem 1 assumes that George’s bound $I_n = f_n^{(0)}$ is known. Thus, this latter must be calculated before $I_n^{(n)}$ can be obtained.

After sorting tasks according to non-decreasing relative deadlines, the algorithm of figure 1 computes iteratively the feasibility bound of theorem 1 for the first $k$ tasks, where $1 \leq k \leq n$. For the reason that it generally delivers better results, we begin calculating $I_k^{(k)}$ from the task with the longest deadline $T_k$ on. This proceeding affects in no way the validity of theorem 1, because this latter makes no assumption on tasks’ order.

In every iteration of the nested for-loop, $I_k^{(k-i+1)}$ is calculated and compared to $d_k$. If for any step $I_k^{(k-i+1)} \leq d_k$ holds, the first $k$ tasks will be feasible because of lemma 5 and of taking into account that theorem 1 guarantees that $I_k^{(k-i)} \geq I_k^{(k)}$ holds for $1 \leq i \leq k$. In this latter case, the algorithm continues running towards $k = n$. However, if $i = 1$ is reached, i.e. $I_k^{(k-i+1)} = I_k^{(k)}$ is reached, and $I_k > d_k$ holds, we would not be able to make any assertion about the feasibility of the first $k$ tasks, so that we will have to consider them as not feasible.

6.2. Feasibility test with $O(n \log n)$

A complexity $O(n \log n)$ is possible by limiting the nested for-loop in figure 1 to certain number of iterations. We propose to fix these iterations to 100, what results in an algorithm we call $ptftnlogn-100$. However, it might be better to choose this value according to particular cases. We denote these similar algorithms by $ptftnlogn-x$. Figure 2 shows necessary changes on $ptftn^2$ in order to obtain the less complex $ptftnlogn-100$.

7. Experimental results

In this section, we present and evaluate some experiments comparing the proposed algorithms with already known polynomial-time feasibility tests. Furthermore, we extended this comparison to include the fastest exact feasibility test, called $AllApprox$, presented in [1]. Mentioned algorithms are contrasted with respect to accuracy and their running time. In order to achieve a sensible comparison, random task sets were uniformly generated for different processor utilizations as recommended in [3]—UUniFast was applied.

Proof: Immediate from lemma 1 and lemma 4.

7.1. Performance comparison

Figure 3 illustrates the performance, in terms of accepted task sets, versus processor utilization. To obtain task parameters, we proceeded as follows: Once generated $u_i$ as mentioned above, we created periods $p_i$ also in a random way with uniform distribution; consequently, we got $e_i = u_i \cdot p_i$. Relative deadlines $d_i$ were uniformly chosen from the range $[e_i, p_i]$. Additionally, we sampled the utilization axis in 2.5% hops, for which the huge amount of 10000 different task sets were generated each time. In this manner, different curves were created for 5, 10, 100, 500 and 1000 tasks per task set. These curves were averaged together in figure 3 in order to show how feasibility tests behave independently of the number of tasks $n$.

As depicted in figure 3, $ptftn^2$ performs almost as well as exact algorithms in the utilization range $(0, 85\%)$. This algorithm outperforms all known polynomial-time tests for utilizations in $(60\%, 90\%)$, reaching a peak at $U = 0.8$ of about 40% more accepted task sets over Devi’s test.

On the other side, $ptftnlogn-100$ accepts around 10% more task sets than Devi’s test in the utilization range $(70\%, 90\%)$. As expected, the performance of this latter algorithm degrades to that of Devi’s test as $n$ increases.

Figure 4 shows a performance comparison versus gap.
\[ U_k = r_k = 0; \]
\[ \text{iters} = 100; \]
\[
\text{sort according to non-decreasing } d_i; \\
\text{for } k = 1 \text{ to } n /* n = \text{number of Tasks in } \tau */ \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text{:} \\
\text:{Figular 2. Algorithm ptftnlogn-100

7.2. Comparison of running time

We implemented algorithms on Matlab—besides using Matlab’s sortrows for sorting task sets where necessary, routine testlist.add, for adding elements to a sorted list, in AllApprox was implemented with \( O(\log n) \). Algorithms were timed on a 2GHz Intel Core 2 Duo machine running Windows XP. For creating curves, no distinction was made between feasible and unfeasible task sets.

Figure 6 compares the average running time versus utilization. Here again, we proceeded as described above in order to generate curves independently of the number of tasks. As it can be observed, proposed algorithms are around two orders of magnitude faster than AllApprox in the utilization interval \([0, 70\%]\), but they are almost two orders of magnitude slower than the density test. In \((70\%, 90\%)\), the running time of ptftn\(^2\) gets drastically closer to Allapprox, but there is still approximately an order of magnitude between them. This drastic change experienced by ptftn\(^2\) responds to the fact that the feasibility bound \( I_k^{(k)} \) of theorem 1 is calculated as it gets necessary—if \( I_k^{(k-i+1)} \leq d_k \) holds for any \( i \), the calculation of \( I_k^{(k)} \) will be broken off, see figure 1—and as expected, we need to calculate the whole \( I_k^{(k)} \) for higher
remains an order of magnitude faster than AllApprox as to reach curves that are relatively independent of
ing exactly as already described. The running time of
8. Conclusions
time algorithms.
In this case, algorithms behave similarly to figure 6.
On its side, figure 8 depicts the running time of algo-
rithms for $U = 0.8$ as the number of tasks grows. The
number of tasks was incremented in 50-task steps proceed-
ing exactly as already described. The running time of $ptfn^2$
remains an order of magnitude faster than AllApprox as $n$
grows, but it gets worse in relation to the other polynomial-
time algorithms.

8. Conclusions

In this paper, two better polynomial-time sufficient fea-
sibility tests were presented for EDF considering arbitrary
relative deadlines and a preemptive uniprocessor schedul-
ing. For complexity $O(n^2)$, we proved that our proposed
test $ptfn^2$ outperforms Devi’s condition in the sense of ac-
curacy. Moreover, we showed that $ptfn^2$ behaves almost
like exact feasibility tests in terms of accepted task sets and
for the utilization range $(0, 85\%)$. A complexity $O(n \log n)$
was also shown to be possible, however, the performance
of our suggested test $ptfnlogn-100$ deteriorates towards the
one of Devi’s test as the number of tasks $n$ grows. As stated
in section 6.2, it is additionally possible to design other tests
with complexity $O(n \log n)$ that might be more convenient
for particular applications.

Finally, considering that $ptfn^2$ performs well when pro-
cessor utilization is less than 90\%, we believe it to be the
best alternative to more time-consuming exact algorithms in
many practical situations. In addition, as the running time is
kept moderate for most practical application, e.g. $n \approx 600$, we
believe $ptfn^2$ also to be more recommendable than the
other polynomial-time feasibility tests.

References

real-time systems with edf scheduling. Proceedings of the
DATE 05 Conference, March 2005.
hard-real-time sporadic tasks on one processor. Proceedings of
schedulability tests. Real-Time Systems, 30(1-2):129–154,
2005.
periodic task systems. Proceedings of the 15th Euromicro
[5] L. George, N. Rivierre, and M. Spuri. Preemptive and non-
preemptive real-time uniprocessor scheduling. Rapport de
Recherche RR-2966, INRIA, 1996.
gramming in hard real-time environments. Journal of the
1996.
Deadline Scheduling for Real-Time Systems: EDF and Re-