An Exact Schedulability Test for Global FP Using State Space Pruning

Artem Burmyakov*
Enrico Bini
Eduardo Tovar*

*CISTER Research Center
CISTER-TR-150503

1, Jan, 2015
Artem Burmyakov*, Enrico Bini, Eduardo Tovar*

*CISTER Research Center
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340509, Fax: +351.22.8321159
E-mail: armbv@isep.ipp.pt, emt@isep.ipp.pt
http://www.cister.isep.ipp.pt

Abstract

We propose an exact schedulability test for sporadic real-time tasks with constrained deadlines, scheduled by Global Fixed Priority (GFP). Our test is faster and less memory consuming than other state-of-the-art exact tests. We achieve such results by employing a set of techniques that cut down the state space of the analysis, which extend the prior work by Bonifaci and Marchetti-Spaccamela. Our test is implemented in C++ code, and it is publicly available.
An Exact Schedulability Test for Global FP
Using State Space Pruning

Artem Burmyakov
CISTER/INESC-TEC, ISEP,
Polytechnic Institute of Porto,
Portugal

Enrico Bini
Scuola Superiore Sant’Anna,
Pisa, Italy

Eduardo Tovar
CISTER/INESC-TEC, ISEP,
Polytechnic Institute of Porto,
Portugal

ABSTRACT

We propose an exact schedulability test for sporadic real-time tasks with constrained deadlines, scheduled by Global Fixed Priority (GFP). Our test is faster and less memory consuming than other state-of-the-art exact tests. We achieve such results by employing a set of techniques that cut down the state space of the analysis, which extend the prior work by Bonifaci and Marchetti-Spaccamela. Our test is implemented in C++ code, and it is publicly available.

1. INTRODUCTION

The key property of a real-time system is time predictability: the operations in such a system must be performed within a strictly defined time. A scheduler is employed to allocate the available CPU time to pending jobs of real-time tasks, using certain priority rules.

At design time, a schedulability test is used to ensure that all system deadlines will be met at runtime. Although fast exact schedulability tests exist for uniprocessor platform [1, 2, 3], existing exact tests for a multiprocessor platform are highly time- and memory-consuming. Hence, a considerable amount of research has focused on sufficient tests, which however significantly overestimate the system demand for processing capacities.

To understand the need for an exact test, we thoroughly evaluated the performance of the sufficient test of Guan et al. [4] against the exact test of Bonifaci and Marchetti-Spaccamela [5], and in certain cases, its pessimism exceeds 50%: more than half of those task sets, reported by Guan’s test as unschedulable, are in fact schedulable. However, due to high computation time and memory consumption, Bonifaci’s exact test becomes intractable even for small systems.

In this paper we derive a faster exact schedulability test for sporadic tasks with constrained deadlines.

1.1 Related works

The first exact schedulability test has been proposed by Baker and Cirinei [6], for several discrete-time schedulers. To check whether a given system is schedulable, the authors solve a reachability problem in a finite state transition graph: the algorithm traverses such a graph until it either finds a state with a violated deadline, or all feasible states are confirmed schedulable.

Bonifaci and Marchetti-Spaccamela [5] improved significantly Baker’s test [6]. They have also refined the complexity bounds for the exact test, showing that it has polynomial space complexity, rather than exponential, as reported in [6]. An efficient C++ implementation for Bonifaci’s test is publicly available. As our work strongly relies on [5], Section 3 will provide more details on that work.

Another exact test was proposed by Geeraerts et al. [7], using formal verification methods. Both Bonifaci’s test [5] and Geeraerts’ test [7] apply to most of online discrete-time schedulers, such as GFP and GEDF, and allow tasks with arbitrary deadlines. Sun and Lipari [8] have instead derived a test specifically for GFP, by using a linear hybrid automaton.

Finally, Guan et al. [9] proposed a test for strictly periodic tasks, for fixed-priority scheduling, using model-checking.

Our evaluation has shown that Bonifaci’s test [5] for GFP is faster, when compared to the exact tests using a timed automaton, than is by Geeraerts et al. [7], and Sun and Lipari [8]. Such a conclusion is based on comparing running times reported in [7], [8] against our evaluation reported in Section 5. The runtime gain of Bonifaci’s test increases noticeably for task sets with a larger number of tasks, and a larger range of task periods. For example, while Geeraerts’ test [7] is constrained to task periods not exceeding 6-8, Bonifaci’s test can deal with larger task periods up to 40. We have also made some initial evaluation of constraint programming and global optimization methods (such an optimization problem can be formulated through the notation proposed in Section 2), but the resulted runtime was much longer than for Bonifaci’s test [5]. For these reasons, we have chosen Bonifaci’s test [5] as an initial ground to apply our improvements. Anyway, we remark that all runtime reduction techniques derived in this work can be applied to any other existing exact test for GFP.

1.2 Contributions of this paper

First, we estimate the pessimism of sufficient schedulability tests for GFP, and confirm the need for a better solution. Then, we derive an exact test, faster than Bonifaci’s test [5], by exploiting (i) a constraint to maximize job interference, (ii) a sufficient schedulability constraint, (iii) a constraint for critical job release instants, and (iv) an optimized clock transition between checked states.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

RTNS ’15, November 4-6, 2015, Lille, France
Copyright 2015 ACM 978-1-4503-3591-1/15/11 ...$15.00.

http://www.iasi.cnr.it/~vbonifaci/software.php
2. DEFINITIONS

We model a real-time system by a set of sporadic tasks \( T = \{ \tau_1, \ldots, \tau_n \} \), wherein each task \( \tau_i = (C_i, D_i, R_i) \) is characterized by an execution time \( C_i \), a relative deadline \( D_i \), and a minimum interarrival time \( P_i \). We consider constrained deadlines \( D_i \leq P_i \). Each task \( \tau_i \in T \) generates a potentially infinite sequence of jobs, whose releases are separated by at least \( P_i \). A job released by \( \tau_i \) has an execution requirement of \( C_i \) time units, and a deadline at \( D_i \), time units after the job arrival time. All task parameters are assumed to be integers. Tasks are scheduled by Global Fixed Priorities (GFP) upon \( m \) identical processors, and are sorted by decreasing priorities. We consider that scheduling decisions are taken at discrete time instants \( N = \{0, 1, 2, \ldots \} \).

To represent the possible scenarios of job releases, we define the release sequence \( R \) as a set of \( n \) functions
\[
R = \{ r_1(t), \ldots, r_n(t) \mid t \in N \},
\]
wherein each function \( r_i : N \to \{0, 1\} \) is such that \( r_i(t) = 1 \) if \( \tau_i \) releases a job at time \( t \), and \( r_i(t) = 0 \) otherwise.

The release sequence \( R \) is said legal if the constraint on the job minimum interarrival times is met; that is:
\[
\forall i = 1, \ldots, n, \forall t, t' \in N, t < t', \quad (r_i(t) = 1 \land r_i(t') = 1) \implies t' - t \geq P_i. \tag{1}
\]

Also, we define the finishing sequence \( F \) as a set of \( n \) functions
\[
F = \{ f_1(t), \ldots, f_n(t) \mid t \in N \},
\]
wherein each function \( f_i : N \to \{0, 1\} \) is such that \( f_i(t) = 1 \) if a job of \( \tau_i \) is completed at time \( t \), and \( f_i(t) = 0 \) otherwise.

Set \( Q \) is defined by
\[
Q = \{ q_1(t), \ldots, q_n(t) \mid t \in N \},
\]
wherein each function \( q_i : N \to \{0, 1\} \) indicates if \( \tau_i \) has a pending job at time \( t \) (in the run queue), defined by:
\[
q_i(t) = \sum_{t'=0}^{t} r_i(t') - \sum_{t'=0}^{t} f_i(t'). \tag{2}
\]

A schedule is represented by a set \( S \) of \( n \) functions
\[
S = \{ s_1(t), \ldots, s_n(t) \mid t \in N \},
\]
with \( s_i(t) = 1 \) if any processor among the \( m \) available ones is allocated to \( \tau_i \) over time \([t, t+1)\), and \( s_i(t) = 0 \) otherwise. With these notations, GFP schedule \( S \) is formally defined by
\[
s_i(t) = 1 \iff q_i(t) > 0 \land \sum_{t'=t}^{t-1} q_i(t') < m, \tag{3}
\]
and the indicator function \( f_i(t) \) of the finishing time, as
\[
f_i(t) = 1 \iff \exists t_r < t : r_i(t_r) = 1 \land \sum_{t'=t_r}^{t-1} s_i(t') = C_i \land s_i(t - 1) = 1. \tag{4}
\]

Fig. 1 illustrates an example of a GFP schedule of \( T = \{ \tau_1, \tau_2, \tau_3 \} \). Below the schedule, we list the respective values for \( R, F, Q, \) and \( S \), where the \( i \)-th row corresponds to task \( \tau_i \), and the \( j \)-th column corresponds to time instant \( t = j - 1 \).

\[
\begin{array}{c|c|c|c|c|c|c|c}
\hline
\text{Notation:} & \text{job release} & \text{job completion} & \text{job preemption} & \text{resource allocation} \\
\hline
R: & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
& 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
& 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
F: & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
& 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
& 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
Q: & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
& 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
& 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\hline
S: & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
& 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
& 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
\end{array}
\]

Time instants \( t_r, t_e \) are said to be the release and completion times of the same job of \( \tau_i \), if it holds:
\[
r_i(t_r) = 1 \land f_i(t_e) = 1 \land \sum_{t=0}^{t_e} r_i(t) = \sum_{t=0}^{t_e} f_i(t), \tag{5}
\]
meaning that i) some job of \( \tau_i \) is released at time \( t_r \), ii) some job of \( \tau_i \) is completed at time \( t_e \), and iii) the number of \( \tau_i \) releases over time \([0, t_e)\) equals to the number of \( \tau_i \) completions over time \([0, t_e)\).

We define schedulability of \( T \) as follows.

DEFINITION 1 (Schedulability of \( T \)). Let \( LR \) denote all legal release sequences of task set \( T \), satisfying (1). \( T \) is said schedulable upon \( m \) processors, if for any \( R \in LR \), all jobs of task \( \tau_i \), \( i = 1, \ldots, n \), meet their deadlines:
\[
\forall R \in LR, \forall i = 1, \ldots, n, \forall (t_r, t_e), \ t_e - t_r \leq D_i, \tag{6}
\]
where \( t_r, t_e \) are the respective release and completion times for the same job of \( \tau_i \), defined by (5).

3. BACKGROUND ON EXACT SCHEDULABILITY TESTS

Bonifaci and Marchetti-Spaccamela [5] analyzed the schedulability of sporadic tasks by traversing a finite non-deterministic state transition graph, searching for a state with a violated deadline. As our work aims at improving their approach, next we revisit the main ideas behind Bonifaci’s work [5].

At a given time \( t \), the state of the set of tasks is modeled by
\[
(c_i, d_i, p_i)_{i=1}^n \in \mathbb{N}^m, \tag{7}
\]
where \( c_i \in \{0, \ldots, C_i\} \) is the remaining execution time of \( \tau_i \) pending job at \( t \), if any; \( d_i \in \{0, \ldots, D_i\} \) is the remaining time until its deadline; and \( p_i \in \{0, \ldots, P_i\} \) is the remaining time until the earliest release of the next job of \( \tau_i \).

A state transition graph for \( T \) is constructed as follows (see also Fig. 2). Each state in the graph represents a system
Table 1: Bonifaci’s test: Task set example

<table>
<thead>
<tr>
<th>index</th>
<th>C_i</th>
<th>P_i</th>
<th>D_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Algorithm 1 Exact schedulability test

1: procedure EXACT_SCHEDULABILITY_TEST
2: \( V \leftarrow \emptyset \) \quad \triangleright \text{initialize } V
3: \( G \leftarrow (0, 0, 0)_{i=1}^{n} \) \quad \triangleright \text{initialize } G
4: while \( G \neq \emptyset \) do
5: \( g \leftarrow \text{Dequeue}(G) \)
6: compute \( G' \) for \( g \) \quad \triangleright \text{from Eq. (8)}
7: for each \( g' \in G' \) do
8: \( \text{if } g' \notin V \text{ then return } \text{Unschedulable} \)
9: \( \text{if } T_i(10) \text{ holds then } \text{deadline miss} \)
10: \( \text{return } \text{Unschedulable} \)
11: end if
12: \( V = V \cup \{g'\} \)
13: \( G = \text{Enqueue}(G, g') \)
14: end for
15: end while
16: return Schedulable
18: end procedure

state \((c_i, d_i, p_i)_{i=1}^{n}\) at a given time \( t \). The initial state is \((0, 0, 0)_{i=1}^{n}\), meaning that no job has been yet released.

The state transition law, which governs the transition from state \( g = (c_i, d_i, p_i) \) at time \( t \) to the next state \( g' = (c'_i, d'_i, p'_i) \) at time \( t + 1 \), is the following:

\[
g' \in G' \iff \begin{cases} c'_i = c_i - s_i(t) + r_i(t + 1) C_i \\ d'_i = \max(d_i - 1, 0) + r_i(t + 1) D_i \\ p'_i = \max(p_i - 1, 0) + r_i(t + 1) P_i, \end{cases} \tag{8}
\]

where \( G' \) is a set of all successors for \( g \) at time \( t + 1 \). In the equation above, \( s_i(t) \) is the schedule function of \( \tau_i \) uniquely determined by the system state \( g \) through (3), and \( r_i(t + 1) \) represents the release function (the “input” to the system) satisfying (1); that is

\[
p_{i} - 1 > 0 \quad \Rightarrow \quad r_{i}(t + 1) = 0
\]

\[
p_{i} - 1 = 0 \quad \Rightarrow \quad r_{i}(t + 1) \in \{0, 1\}. \tag{9}
\]

State \((c'_i, d'_i, p'_i)_{i=1}^{n}\), at time \( t \), is a scheduling failure state if some job misses its deadline:

\[
c'_i - r_{i}(t) C_i > d'_i - r_{i}(t) D_i, \tag{10}
\]

with \( r_{i}(t) \) defined by (9). This condition is true when the remaining execution time for a job (LHS of (10)) exceeds the remaining time until its deadline (RHS of (10)). \( r_{i}(t) C_i, r_{i}(t) D_i \), are subtracted to correctly consider the case when the job deadline of \( \tau_i \) coincides to the next release of \( \tau_i \).

Once the scheduling failure state is encountered, the algorithm reports unschedulability of \( T \), and terminates. If instead all feasible states have been checked, and no failure state has been detected, then \( T \) is reported schedulable.

Algorithm 1 implements Bonifaci’s test using breadth-first search. It maintains two additional data structures: \( V \) is a set of checked states at previous iterations, and \( G \) is a FIFO queue, containing states for further examination. The set \( V \) is initially empty, and queue \( G \) contains only the initial state \( g_0 = (0, 0, 0)_{i=1}^{n} \). At the first iteration of the while loop, the algorithm removes \( g_0 \) from \( G \), and computes set \( G' \) of successors for \( g_0 \) using (8). Then, each state \( g' \in G' \) is checked for a deadline miss (line 9), added to a list of checked states \( V \) (line 12), and added to queue \( G \), to examine the \( g' \) successors at further iterations. At each iteration of the while loop, the algorithm removes the first state \( g \) from queue \( G \) (line 5), and computes the set \( G' \) of successors for \( g \) (line 6). Each state \( g' \in G' \) that has not been checked yet (that is \( g' \notin V \)), is checked for a deadline miss, and added to \( V \) and \( G \). The algorithm terminates when \( G \) becomes empty, meaning that all feasible system states were examined.

A higher runtime efficiency of Bonifaci’s Algorithm 1 compared to other exact tests is mainly thanks to condition in line 8: each system state in a graph is checked only once. However, the same approach cannot be applied directly to tests using a timed automaton, due to specific of a timed automaton. One solution has been proposed by Geeraerts et al. [7], who derived a so called simulation relation technique for a timed automaton, but it does not seem to be more efficient than Bonifaci’s approach.

Anyway, Algorithm 1 has exponential time and polynomial space complexity, and it might not terminate in a reasonable time even for a small \( T \). According to our evaluation, Bonifaci’s test is capable to deal with up to 5–6 tasks scheduled upon 2 processors, considering very small range of task periods, not exceeding 40.

Consider \( T \) with the parameters reported in Table 1, to be scheduled by GPF upon \( m = 2 \) processors. In Fig. 2 we report fragments of the state transition graph for each \( T \). The total number of states in the full graph is 191.

For the same \( T \), our test checks 12 states only, instead of 191, in one sixth of the running time relative to [5], and the efficiency of our test increases for larger task sets.

4. A FASTER SCHEDULABILITY TEST

In this section we derive a faster exact schedulability test for \( T \). Below we test schedulability of \( T_k \), assuming that \( T_1, \ldots, T_{k-1} \) are schedulable.
4.1 Job interference

We first show that, when analyzing the schedulability of \( \tau_k \), we can safely ignore any job of a higher-priority task \( \tau_i \), \( i < k \), that causes no interference to any lower-priority tasks \( \tau_{i+1}, \ldots, \tau_k \).

Let us define job interference as follows.

**Definition 2 (Job interference).** Let \( J_i \) denote an arbitrary job of \( \tau_i \), with release and completion times denoted by \( t_r \) and \( t_c \), respectively. Job \( J_i \) is said to interfere with a lower priority job \( J_k \) of \( \tau_i \), \( i > k \), if at some time \( t \) \( \in [t_r, t_c] \) a processor is allocated to \( J_i \), but not to \( J_k \):

\[
\exists \ell > i, \ \exists q \in [t_r, t_c]: \ s_i(t) = 1 \land q_i(t) = 1 \land s_k(t) = 0, \ \text{(11)}
\]

with \( s_i(t), q_i(t) \) defined by (3) and (2), respectively.

We clarify this definition on an example. Let \( T = \{\tau_1, \ldots, \tau_k\} \) be scheduled upon \( m = 2 \) processors. Consider the release sequence \( R \) for \( T \) as depicted in Fig. 3a, and suppose that we analyze schedulability of task \( \tau_k \). Job \( J_{k-1} \) of \( \tau_k \) interferes with job \( J_{k-1} \) of \( \tau_k \) at time \( t = 2 \), as (11) holds:

\[
s_1(2) = 1 \land q_2(1) = 1 \land s_2(2) = 0.
\]

Instead, job \( J_{k-1} \) does not interfere with \( J_{k-1} \), as (11) is violated. By removing such non-interfering jobs, we can produce a different arrival sequence that does not affect the schedule of task \( \tau_k \). Let us transform \( R \) into \( R' \), by erasing all jobs of task \( \tau_i \), \( i < k \), which do not interfere with lower priority jobs. These jobs are \( J_{k-2}, J_{k-1} \), and \( J_{k-1} \). Observe that the amount of resource available for \( \tau_k \) in \( R' \) is the same as in \( R \).

**Theorem 1.** Assume that \( \tau_1, \ldots, \tau_{k-1} \) are schedulable. Let \( R = \{r_1(t), \ldots, r_k(t)\} \) be any legal release sequence for \( T \), and let \( J_{i,t} \) denote the \( \tau_i \) job, released at time \( t \). Let \( R' \) be a new release sequence that excludes all jobs of task \( \tau_i \) from \( R, i < k \), which violate the interference condition (11):

\[
R' = \{r'_1(t), \ldots, r'_k(t)\}:
\]

\[
r'_i(t) = \begin{cases} 1, & \text{if } r_i(t) = 1 \text{ and } (11) \text{ holds for } J_{i,t} \smallsetminus J_{k-1} \\ 0, & \text{otherwise} \end{cases}
\]

\[
r'_k(t) = r_k(t).
\]

(12)

Then, the job of task \( \tau_k \), pending at time \( t \) (if exists) misses its deadline in \( R \) if it misses its deadline in \( R' \).

The proof of Theorem 1 is provided in the Appendix.

According to Theorem 1, the worst-case release sequence for \( \tau_k \) is among those satisfying the following condition.

**Corollary 1.** Let \( R_{T, \text{reduced}} \) denote all legal release sequences for \( T \), wherein each job of \( \tau_i \), \( i = 1, \ldots, k-1 \), interferes with a lower priority job, as defined by (11). \( \tau_k \) is schedulable for each legal \( R \), satisfying (1), iff \( \tau_k \) is schedulable for each \( R' \in R_{T, \text{reduced}} \).

We next apply Corollary 1 to speed-up Algorithm 1. Let us extend state definition (7) at time \( t \) to

\[
(c_i, d_i, p_i, b_i)^k \Bigg|_{i=1}
\]

where \( b_i \) is boolean, such that \( b_i(t) = 1 \) iff \( \tau_i \) has a pending job at \( t \), and that job has interfered with a lower priority one by time \( t \) inclusive (meaning that condition (11) holds for that \( \tau_i \) job at some time \( t' \leq t \)).
Constraint (14) allows to totally eliminate the existence of all non-interfering jobs from release scenarios. Later, in Section 5, we report the evaluation results, which confirm the high efficiency of (14) in pruning the state space; that efficiency significantly increases with the number of tasks $n$.

### 4.2 Sufficient schedulability condition

Next, we prune the state space $G'$ for Algorithm 1 through applying a sufficient schedulability condition. In fact, if a sufficient schedulability condition holds for some system state, then this state cannot lead to a failure state, and thus we do not need to examine its successors.

At time $t$, let state $(c_i, d_i, p_i)_{i=1}^m$ be such that $c_k > 0$, meaning that $\tau_k$ has a pending job at time $t$, with remaining execution time $c_k$ and a deadline at time $t + d_k$.

Applying Baruah's analysis [10], the amount of resource allocated to $\tau_k$ over time $[t, t + d_k]$ is at least $d_k - I_k$, (15)

![Figure 4: Pruned state transition graph by (14)](image)

Figure 5: Maximum job interference

![Figure 5: Maximum job interference](image)

Table 2: Critical release instant: Task set example

<table>
<thead>
<tr>
<th>$i$</th>
<th>$C_i$</th>
<th>$P_i$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

where $I_k$ is the upper bound on the interference, caused by $\tau_1, \ldots, \tau_{k-1}$ on $\tau_k$, computed by (see Fig. 5 for intuition)

$$I_k = \frac{W}{m}$$ (16)

with the maximum aggregated workload $W$ for $\tau_1, \ldots, \tau_{k-1}$ computed by

$$W = \sum_{i=1}^{k-1} W_i$$ (17)

$$W_i = \min(c_i, d_i) + \ell_i C_i + \min(C_i, \Delta_i),$$

with

$$\ell_i = \max \left(0, \left[ \frac{d_i - p_i}{P_i} \right] \right)$$

and

$$\Delta_i = d_i - p_i - \ell_i, P_i.$$ (18)

If the lower bound (15) on the supply allocated to $\tau_k$ is not lower than the remaining demand $c_k$ of $\tau_k$, that is $d_k - I_k \geq c_k$, then $\tau_k$ is guaranteed schedulable. Thus, Algorithm 1 needs to check the successor states for $(c_i, d_i, p_i)_{i=1}^m$ only if

$$d_k - I_k < c_k.$$ (19)

We have chosen condition (19) due to its low computation time. However, many other tests could be used instead of (19), and their performance remains to be analyzed.

### 4.3 Critical release instant

Davis and Burns [11] have shown that the worst-case execution scenario for a job of task $\tau_k$ occurs when that job is released at such time $t$ ($r_k(t) = 1$), when all $m$ processors are occupied by higher priority tasks $\tau_1, \ldots, \tau_{k-1}$ (that is, $\sum_{i=1}^{k-1} q_i(t) \geq m$), but there is at least one processor idle during the preceding time interval $[t-1, t]$ (that is, $\sum_{i=1}^{k} q_i(t-1) < m$):

$$r_k(t) = 1 \Rightarrow \sum_{i=1}^{k-1} q_i(t) \geq m \land \sum_{i=1}^{k} q_i(t-1) < m.$$ (20)

Further details can be found in Theorem 1 of [11].

We next adapt such an approach to restrict the release times for $\tau_1, \ldots, \tau_{k-1}$. We first provide an example. Consider $T = \{\tau_1, \ldots, \tau_k\}$ with parameters as reported in Table 2, scheduled upon $m = 2$ processors. We analyze the schedulability of $\tau_k$.

Fig. 6a depicts a legal release sequence for $T$, denoted by $R$. In such $R$, job $J_{1,2}$ of $\tau_1$ causes no interference to other jobs until time 10. Observe also that, after releasing $J_{1,2}$ at time 8, $\tau_1$ cannot release another job until $\tau_1$’s deadline at time 13. Then, we can safely postpone the release of $J_{1,2}$ until time 10, as depicted in Fig. 6b.

The same reasoning does not apply, however, to job $J_{3,1}$ of $\tau_3$: delaying the release of $J_{3,1}$ might potentially affect the release time of $J_{3,2}$ (due to the constraint on the minimal
time separation $P_3$, and $J_{3,2}$ in turn affects the schedulability of $\tau_4$.

We next formalize the argument. For an arbitrary release sequence $R$, let time $t$ be such that:

1. $\tau_2$ has a pending job at time $t$: $q_2(t) = 1$;
2. $\tau_i$, with $i < k$, releases a job at time $t$: $r_i(t) = 1$;
3. $\tau_i$ cannot release another job until $\tau_i$’s deadline:
   \[ P_i - D_i + (t - t_r) \geq 0, \]  
   where $t_r$ denotes the release time for $\tau_i$ job, pending at time $t$.

At such time $t$, there should be at least $m + 1$ pending jobs for $\tau_1, \ldots, \tau_k$:

\[ \forall t, i < k: \begin{cases} q_i(t) = 1 \\ r_i(t) = 1 \end{cases} \Rightarrow \sum_{i=1}^{k} r_i(t) > m. \]  

The efficiency of (22) in proving the state space is higher for lower ratios $P_i/p_{\text{min}}$, with $P_{\text{min}} = \min_{i=1,\ldots,k-1} P_i$; that is due to the presence of (21) in (22).

The release times for $\tau_1, \ldots, \tau_{k-1}$ can be further constrained, by exploring their periods $P_1, \ldots, P_{k-1}$. Recall the release sequence $R'$ as depicted in Fig. 6b. At time 8, no job is released by $\tau_1, \ldots, \tau_3$, although each of them could; the next job is only released 2 time units later, at time 10. To maximize interference on $\tau_4$, we can transform $R'$ into a new $R''$, by shifting 2 time units left all release instants for $\tau_1, \ldots, \tau_3$, occurred after time 8, as depicted in Fig. 6c. Clearly, interference on $\tau_4$ cannot decrease due to such a transformation.

Thus, we require that each $R$ satisfies the condition:

\[ \sum_{i=1}^{k-1} p_i(t) > 0, \quad \forall t \]  

in order to exclude case $\sum_{i=1}^{k-1} p_i(t) = 0$, when each $\tau_1, \ldots, \tau_{k-1}$ can release another job, but none of them does. Due to lack of space, we do not provide a formal proof for constraints (22) and (23). Such a proof can be conducted by analogy to the proof of Theorem 1 in [11], which results in (20).

We conclude that the set $G'$ of successor states, defined by transition law (8), can be pruned further, by adding constraints (20), (22), and (23).

### 4.4 State transition

To avoid unnecessary checks for deadline misses at every time instant, we optimize the clock transition between checked system states as follows.

Let state $c_i, d_i, p_i)_{i=1}^{k}$, at time $t$, be such that $\tau_i$ does not miss any deadline, i.e., $c_i \leq d_i$ by condition (10).

Suppose first that at most $m$ jobs are pending at $(c_i, d_i, p_i)_{i=1}^{k}$; that is, $\sum_{i=1}^{k} \min(c_i, 1) \leq m$ (see Fig. 7a). In this case, no deadline miss can occur until the time when another job can be released, that is time $t + \Delta t$, with $\Delta t$ defined by

\[ \Delta t = \left( \min_{i=1,\ldots,k} p_i \right)_1, \]  

where $(x)_1$ denotes $\max(1, x)$.

Suppose instead that more than $m$ jobs are pending at $(c_i, d_i, p_i)_{i=1}^{k}$ (see Fig. 7b). Let $\Delta t$ denote the remaining time until the next system event might occur for some task $\tau_i$ (job release, completion or deadline):

\[ \Delta t = \left( \min_{i=1,\ldots,k} (c_i, p_i, d_i) \right)_1. \]  

Let $q' = (c_i', d_i', p_i')_{i=1}^{k}$ be a successor state for $q = (c_i, d_i, p_i)_{i=1}^{k}$, $\Delta t$ time units later, with $\Delta t$ defined as above. If any of the intermediate states between $q$ and $q'$ is a failure state, then $q'$ is a failure state as well.

Thus, the optimized clock transition $\Delta t$ between checked system states is computed by (24) and (25):}

\[ \Delta t = \begin{cases} \left( \min_{i=1,\ldots,k} p_i \right)_1, & \text{if } \sum_{i=1}^{k} \min(c_i, 1) \leq m \\ \left( \min_{i=1,\ldots,k} (c_i, p_i, d_i) \right)_1, & \text{otherwise.} \end{cases} \]  

![Figure 7: Clock increment $\Delta t$: cases](image)
The transition law (8) is updated accordingly, by replacing the clock increment “1” in (8) and (9) by $\Delta t$:

$$(c'_i, d'_i, p'_i)_{i=1}^k \in G' \iff$$

$$
\begin{align*}
&c'_i = (c_i - s_i(t) \Delta t) + r_i(t + \Delta t) C_i \\
&d'_i = (d_i - \Delta t) + r_i(t + \Delta t) D_i, \quad i = 1, \ldots, k \\
p'_i = (p_i - \Delta t) + r_i(t + \Delta t) P_i
\end{align*}
$$

(27)

where $g' = (c'_i, d'_i, p'_i)$. $\Delta t$ and $s_i(t)$ are defined by (26), (3), and $r_i(t + \Delta t)$ satisfies (1):

$$
p_i - \Delta t > 0 \Rightarrow r_i(t + \Delta t) = 0
$$

$$
p_i - \Delta t = 0 \Rightarrow r_i(t + \Delta t) \in \{0, 1\}
$$

4.5 Procedure for the schedulability test

The optimized procedure for an exact schedulability test is as follows. In Algorithm 1, we replace the state transition law (8) by (27), and we incorporate the pruning constraints (14), (19), (20), (22), (23) into (27).

5. EVALUATION

We finally evaluate the performance of the exact schedulability test presented in Section 4.5. The test is implemented using C++ libraries, by extending Bonifaci’s tool [5], and its code is publicly available\(^3\).

The experiments are conducted on a hardware platform with the following specifications:

- Processor: Intel Core i7-4710MQ CPU @ 2.5GHz
- Operating memory (RAM): 15,50 GB 1600 MHz
- System type: 64-bit
- Operating system: Ubuntu 14.04 LTS

5.1 Task set generation

Sporadic task sets $T = \{\tau_i = (C_i, P_i)\}$ with implicit deadlines $D_i = P_i$ are randomly generated by specifying the number of tasks $n$, the total task set utilization $U_T$, the maximum individual task utilization $U_{\text{max}}$, and the range for task periods $[P_{\text{min}}, P_{\text{max}}]$.

The minimum period $P_{\text{min}}$ is randomly taken from the range $[3; 10]$, and all the task periods are generated such that the specified ratio $P_{\text{min}}/P_{\text{max}}$ holds. Task execution times $C_i$ are chosen by solving the following linear integer optimization problem, using CPLEX:

$$
\text{minimize } U_T - \sum_{i=1}^{n} \frac{C_i}{P_i} + |U_{\text{max}} - C_i/r_i| \\
\text{subject to } \begin{align*}
0 < C_i < P_i, & \quad i = 1, \ldots, n \\
U_T - \sum_{i=1}^{n} \frac{C_i}{P_i} & \leq \delta U_T \\
|U_{\text{max}} - C_i/r_i| & \leq \delta U_{\text{max}} \\
C_i/r_i & \leq C_i/r_i, \quad i = 1, \ldots, n,
\end{align*}
$$

where $C_i, i = 1, \ldots, n$, are integer optimization variables, $|x|$ denotes the absolute value of $x$, and index $i^*$ corresponds to task $\tau_{i^*}$ having the maximum utilization $U_{\text{max}}$.

\(^3\)www.cister.isep.ipp.pt/docs/CISTER-TR-150503

Table 3: Key parameters: default values

<table>
<thead>
<tr>
<th>Settings</th>
<th>Table 3: Key parameters: default values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of processors, $m$</td>
<td>2</td>
</tr>
<tr>
<td>Number of tasks, $n$</td>
<td>5</td>
</tr>
<tr>
<td>Task set utilization, $U_T$</td>
<td>1.6</td>
</tr>
<tr>
<td>Maximum individual task utilization, $U_{\text{max}}$</td>
<td>0.6</td>
</tr>
<tr>
<td>Minimum task period, $P_{\text{min}}$</td>
<td>[3, 10]</td>
</tr>
<tr>
<td>Ratio between the maximum and minimum task periods, $P_{\text{max}}/P_{\text{min}}$</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 8: Runtime of exact tests (logarithmic scale)

\(i^*\) is randomly taken from range $[1, \ldots, n]$. We allow a relative deviation $\delta U_T = 1.5\%$ between the specified value $U_T$ and the actual tasks utilization $\sum C_i/r_i$, as well as deviation $\delta U_{\text{max}} = 2.5\%$ between the specified value $U_{\text{max}}$ and the actual maximum task utilization $\max_{i=1,\ldots,n} C_i/r_i$.

We randomly generate task sets for the settings reported in Table 3. In each experiment, one key parameter varies, while the rest are left equal to the default values in Table 3.

The choice for parameter values is constrained by a hardware limitation of 16 GB of operating memory. $U_T$ is chosen such that the pessimism of Guan’s test [4] is maximized (that is the case when the usage of an exact test makes more sense). For a thorough evaluation, the pessimism of Guan’s test is analyzed for varying $U_T$ as well.

5.2 Runtime reduction

We first compare the runtime of our test against Bonifaci’s test [5]\(^4\), for $m = 2$. Figs. 8a, 8b report the average runtime for a varying number of tasks $n$, and a varying ratio $P_{\text{max}}/P_{\text{min}}$ of task periods, considering only schedulable task sets. We confirm a significant runtime reduction for our test, allowing task sets with larger $n$ and $P_{\text{max}}/P_{\text{min}}$. However, the runtime complexity remains exponential in $n$.

The second experiment is conducted for $m = 3$. The

\(^4\)We speeded-up the original code available at http://www.iiasi.cnr.it/~vbonifaci/software.php by a factor 10–20, by recompiling it at optimized settings.
average runtime for our test is reported in Figs. 8c, 8d. For such settings, Bonifaci’s test requires more than 16 GB of memory in most cases, so that the comparison to our test is infeasible. In 5% of cases, our test exceeds 16 GB as well, and those cases are discarded.

Figs. 9a-9c report the contribution of each pruning constraint into the state space reduction. The plotted size reduction is computed by

$$\text{reduction}_{(x)} = \frac{N_{(x) \text{ excluded}}}{N},$$

where $N$ is the number of states checked by the algorithm employing all pruning constraints (14), (19), (20)-(23), (27), and $N_{(x) \text{ excluded}}$ is the number of states checked by the same algorithm excluding the pruning constraint $(x)$.

Despite of the polynomial space complexity, our evaluation shows that the required system memory for our algorithm (as for all other existing exact tests) increases significantly with the number of tasks and their periods. For example, for task sets comprised of just 10 tasks, and their periods not exceeding 40, the required memory already exceeds 16 GB in most cases.

5.3 Comparison of exact and sufficient tests

To motivate the need for an exact test, we also evaluate the pessimism of Guan’s sufficient test [4], which in turn outperforms most of other existing sufficient tests$^5$. The experiments are conducted when varying $n$ and $U_T$, for both $m = 2$ and $m = 3$. The results are reported in Figs. 10a-10d. We confirm a significant pessimism of Guan’s test, exceeding 50% under certain settings. However, our evaluation is limited to very small task sets, due to the high memory consumption of the exact test.

6. CONCLUSION

We evaluated the performance of sufficient multiprocessor schedulability tests for GFP, and confirmed their significant pessimism. Then, to overcome the major drawbacks of these existing exact tests, which is high computation time and memory consumption, we derived an improved exact test, by extending Bonifaci and Marchetti-Spaccamella’s work [5]. Evaluation confirmed high efficiency of proposed improvements, although the algorithm remains exponential in time and polynomial in space.

7. ACKNOWLEDGMENTS

This work was partially supported by FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER); also by FCT/MEC and the EU ARTEMIS JU within projects ARTEMIS/0003/2012 - JU grant 333053 (CONCERTO) and ARTEMIS/0001/2013 - JU grant 621429 (EMC2); also by FCT/MEC and ERDF through COMPETE (Operational Programme ’Thematic Factors of Competitiveness’), within project FCOMP-01-0124-FEDER-020536 (SMARTS).

8. REFERENCES

Theorem 1. Let S and S’ have the property:

\[ \forall i, t : q_i^\prime(t) = 1 \rightarrow s_i(t) = s_i^\prime(t), \] (28)

where \( q_i(t) \) is defined by (2), and \( s_i(t) \) and \( s_i^\prime(t) \) are defined by (3), for \( S \) and \( S' \) respectively.

Observe that Lemma 1 does not require feasibility of \( T \).

To prove Lemma 1, we explore definition (3) of supply function \( s_i(t) \): a job gets supply over time \([t, t+1)\) (that is \( q_i(t) = 1 \) and \( s_i(t) = 1 \)), if the number of higher priority jobs at \( t \) is less than \( m \) (that is \( \sum_{t=1}^{i-1} q_i(t) < m \)):

\[ s_i(t) = 1 \Leftrightarrow \begin{cases} q_i(t) = 1 \\ \sum_{t=1}^{i-1} q_i(t) < m \end{cases} \] (29)

\[ s_i(t) = 0 \Leftrightarrow \begin{cases} q_i(t) = 0 \\ q_i(t) = 1 \\ \sum_{t=1}^{i-1} q_i(t) \geq m \end{cases} \] (30)

\forall i, t

where \( q_i(t) \) denotes logical OR.

Also, the necessary constraints for values of \( s_i(t) \) are:

\[ s_i(t) = 1 \Rightarrow \sum_{t=1}^{i-1} q_i(t) < m, \] (31)

\[ s_i(t) = 0 \wedge q_i(t) = 1 \Rightarrow \sum_{t=1}^{i-1} q_i(t) \geq m, \] (32)

\forall i, t

We are now ready to prove the lemma.

Proof. We prove the lemma by contradiction. Suppose that (28) is violated for some task \( \tau \), at time \( t^* \), that is

\[ \exists i^*, t^* : q_{i^*}^\prime(t^*) = 1 \wedge s_{i^*}(t^*) \neq s_{i^*}(t^*). \] (33)

Without loss of generality, let \( t^* \) be the earliest time, when (28) is violated, meaning that

\[ \forall i, t < t^* : q_i(t) = 1 \rightarrow s_i(t) = s_i(t). \] (34)

As \( s_i(t) \) and \( s^\prime_i(t) \) are boolean, (33) might hold in two cases:

\[ \begin{cases} s_{i^*}(t^*) = 1 \\ s^\prime_{i^*}(t^*) = 0 \end{cases} \] (35) or \[ \begin{cases} s_{i^*}(t^*) = 0 \\ s^\prime_{i^*}(t^*) = 1 \end{cases} \] (36)

We next show that both cases are infeasible, meaning that (33) cannot hold.

Case 1: Substituting (31), (32) into (35), we get that

\[ \begin{cases} s_{i^*}(t^*) = 1 \\ s^\prime_{i^*}(t^*) = 0 \end{cases} \] (37)

\[ q_{i^*}^\prime(t^*) = 0 \] (38)

From (37), \( \tau_{i^*} \) has a job at time \( t^* \), in \( S' \). Let \( J_i^* \) denote that job, and let \( t_{i^*} \) denote its release time, such that

\[ r_{i^*}(t_{i^*}) = 1. \] (39)

As \( J_i^* \) has not been completed by time \( t^* \), \( J_i^* \) has not received \( C_i \) units of resource by time \( t^* \):

\[ \sum_{t=t_{i^*}}^{t^*} s_i^\prime(t) < C_i. \] (40)

From (12), whenever \( \tau_{i^*} \) releases a job in \( R' \), it also releases a job in \( R \). Due to (39) and (12),

\[ r_{i^*}(t_{i^*}) = 1 \] (41)

\[ r_{i^*}(t_{i^*}) = 1, \] (42)

that is \( \tau_{i^*} \) releases a job at time \( t_{i^*} \), in \( R \). Let \( J_{i^*} \), denote that job, as well as \( t_{i^*} \) - the completion time for \( J_{i^*} \).

Considering (38), \( \tau_{i^*} \) has no job pending at time \( t^* \), in \( S \), meaning that \( J_{i^*} \) has been completed by time \( t^* \):

\[ t_{i^*} \leq t^* \]

\[ \sum_{t=t_{i^*}}^{t^*} s_i^\prime(t) = C_i. \] (43)
what together with (40) yields contradiction to (34), as
\[
\sum_{t=t_{i^*}^*+1}^{t_{i^*}^*+1} s_{i^*}(t) = C_i \quad \land \quad \sum_{t=t_{i^*}^*+1}^{t_{i^*}^*+1} s'_{i^*}(t) < C_i
\]
implies that
\[
\exists t \in [t_{i^*}^*, t^*): q_{i^*}(t) = 1 \quad \land \quad s_{i^*}(t) \neq s'_{i^*}(t) \quad (41)
\]
Thus, Eq. (35) is infeasible.

Case 2: Suppose that (36) holds. Substituting (29), (30) into (36), we get that
\[
\begin{align*}
& s_{i^*}(t^*) = 0 \quad (38) \\
& s'_{i^*}(t^*) = 1 \\
& q_{i^*}(t^*) = 1 \\
& \sum_{t=1}^{i^*+1} q(t^*) \geq m \\
& \sum_{t=1}^{i^*+1} q'(t^*) < m
\end{align*}
\]
The first case
\[
q_{i^*}(t^*) = 0 \\
q'_{i^*}(t^*) = 1
\]
is infeasible; the proof is conducted by analogy to Case 1, for Eq. (37), (38).

Suppose that the second case holds:
\[
q_{i^*}(t^*) = 1 \\
\sum_{t=1}^{i^*+1} q(t^*) \geq m \\
\sum_{t=1}^{i^*+1} q'(t^*) < m
\]
Due to (43) and (44), there exists such a task \( \tau_{i^*} \), with \( i^* < i^* \), that satisfies the constraints:
\[
\begin{align*}
& q_{i^*}(t^*) = 1 \\
& \sum_{t=1}^{i^*+1} q(t^*) = m \\
& q'_{i^*}(t^*) = 0. \quad (47)
\end{align*}
\]
Let \( J_{i^*} \) denote \( \tau_{i^*} \) job, pending at time \( t^* \) in \( S \) (there is such a job, due to (45)), and let \( t_{i^*} \) denote its release time:
\[
\begin{align*}
& r_{i^*}(t_{i^*}^*) = 1 \\
& \sum_{t=t_{i^*}^*+1}^{t_{i^*}^*+1} s_{i^*}(t) < C_i
\end{align*}
\]
Observe that \( J_{i^*} \) interferes with a lower priority job at time \( t^* \): a processor is assigned to \( J_{i^*} \) at time \( t^* \) (that is due to (45) and (46)), and the number of pending jobs at \( t^* \) exceeds \( m \) (that is due to (42) and (43)). From definition (12) of \( R' \):
\[
\begin{align*}
& \sum_{t=t_{i^*}^*+1}^{t_{i^*}^*+1} s_{i^*}(t) = C_i \\
& (41) \quad \land \quad (42) \quad \Rightarrow \quad r_{i^*}^*(t_{i^*}) = 1,
\end{align*}
\]
meaning that \( \tau_{i^*} \) released a job at time \( t_{i^*}^* \), in \( R' \). Let \( J_{i^*} \) denote that job, and \( t_{i^*}^* \) its completion time. Due to (47), \( \tau_{i^*} \) has no pending job at \( t^* \); in \( S' \), meaning that \( J_{i^*} \) has been completed by time \( t^* \):
\[
\sum_{t=1}^{t_{i^*}^*} s_{i^*}(t) = C_i
\]
what together with (48) contradicts to assumption (34):
\[
\begin{align*}
& \sum_{t=1}^{t_{i^*}^*} s_{i^*}(t) = C_i \\
& (41) \quad \land \quad (42) \quad \Rightarrow \quad r_{i^*}^*(t_{i^*}) = 1,
\end{align*}
\]
meaning that (34) is violated:
\[
\exists t \in [t_{i^*}^*, t^*): q_{i^*}(t) = 1 \quad \land \quad s_{i^*}(t) \neq s'_{i^*}(t)
\]
We conclude that both cases, (35) and (36), are infeasible, and lemma statement (28) holds always.

\[\Box\]

We finally prove Theorem 1.

**Proof for Theorem 1.**

**Necessity:** We first prove that feasibility of \( R \) implies feasibility of \( R' \). The proof is conducted by contradiction. Suppose that exist such \( R, R' \), satisfying (12), that \( R \) is feasible, but \( R' \) is infeasible. For \( R' \), let \( t_{dm} \) denote the earliest missed deadline for \( \tau_k \); consequently, \( t_{dm} - D_k \) is the release time of the job missing the deadline:
\[
r_k(t_{dm} - D_k) = 1, \quad (49)
\]
Due to (12), if \( \tau_k \) has released a job at time \( t_{dm} - D_k \) in \( R' \), then it has also released a job in \( R \):
\[
r_k(t_{dm} - D_k) = 1, \quad (51)
\]
Due to the assumption that \( R \) is feasible,
\[
\sum_{t=t_{dm} - D_k}^{t_{dm} - D_k} s_k(t) = C_k, \quad (52)
\]
what contradicts to (50) due to Lemma 1.

**Sufficiency:** The sufficiency proof shows that feasibility of \( R' \) implies feasibility of \( R \). Such a proof is conducted by analogy to the necessity proof, by swapping functions \( r_k(t) \), \( s_k(t) \) and \( r'_{i^*}(t) \), \( s'_{i^*}(t) \) in Eq. (49)-(52).

The theorem follows.

\[\Box\]