Polynomial-Time Exact Schedulability Tests for Harmonic Real-Time Tasks

Vincenzo Bonifaci · Alberto Marchetti-Spaccamela
Nicole Megow · Andreas Wiese

IASI-CNR Rome · Sapienza University of Rome
TU Berlin · MPII Saarbrücken
Outline

1. Model and Related Work
2. FP-Schedulability
3. EDF-Schedulability
Sporadic Task Model – Constrained Deadlines

Tasks $\tau_1, \tau_2, \ldots, \tau_n$

Task $\tau_i$ has:
- a worst-case computation time $c_i$
- a relative deadline $d_i$
- a minimum inter-arrival time (period) $p_i$

Example:

![Diagram showing time intervals for $c_i$, $d_i$, and $p_i$]

We assume all parameters integral and constrained deadlines ($d_i \leq p_i$)
Scheduling Model

- One processor (so $\sum_i c_i/p_i \leq 1$)
- Preemptions allowed
- No preemption overheads
Preemptive Scheduling Policies Considered

- **Fixed Priority (FP)**
  - task-level priority
  - give priority to the tasks according to a static ordering ($p_i$: Rate Monotonic, $d_i$: Deadline Monotonic, ...)
  - jobs inherit priority from the originating task

- **Earliest Deadline First (EDF)**
  - job-level priority
  - give priority to the jobs with earliest absolute deadlines

In both cases, at any time step, schedule the available job with highest priority
A task system is **ALG-schedulable** if, for every of its (legal) job sequences, algorithm ALG constructs a feasible schedule.
A task system is **ALG-schedulable** if, for every of its (legal) job sequences, algorithm ALG constructs a feasible schedule

Let $P = \max_i p_i$, $n =$ number of tasks

A schedulability test runs in

- **pseudopolynomial time** if it runs in time $O(n^a P^b)$ for some $a, b$
- **polynomial time** if it runs in time $O(n^a (\log P)^b)$ for some $a, b$
A task system is \textit{ALG-schedulable} if, for every of its (legal) job sequences, algorithm ALG constructs a feasible schedule.

Let \( P = \max_i p_i \), \( n \) = number of tasks

A schedulability test runs in
- \textit{pseudopolynomial time} if it runs in time \( O(n^a P^b) \) for some \( a, b \)
- \textit{polynomial time} if it runs in time \( O(n^a (\log P)^b) \) for some \( a, b \)

\textbf{Remark.} Enough to analyze the \textit{Synchronous Arrival Sequence}:
- jobs of \( \tau_i \) released at 0, 1 \cdot p_i, 2 \cdot p_i, 3 \cdot p_i, \ldots \)
FP-schedulability is often verified via Response Time Analysis.
FP-schedulability is often verified via Response Time Analysis.

**Response Time Analysis Problem**

**Input:** priority-ordered task system \( \tau \), integer \( R \)

**Question:** is the response time of \( \tau_n \) at most \( R \)?
FP-schedulability is often verified via Response Time Analysis.

**Response Time Analysis Problem**

**Input:** priority-ordered task system $\tau$, integer $R$

**Question:** is the response time of $\tau_n$ at most $R$?

Liu & Layland (1973) and subsequent work

**Response Time Analysis** can be solved in pseudopolynomial time.
FP-schedulability is often verified via Response Time Analysis.

**Response Time Analysis Problem**

**Input**: priority-ordered task system $\tau$, integer $R$

**Question**: is the response time of $\tau_n$ at most $R$?

Liu & Layland (1973) and subsequent work

Response Time Analysis can be solved in pseudopolynomial time.

Eisenbrand & Rothvoss (2008)

Response Time Analysis is (weakly) NP-hard.
Schedulability Testing: EDF

**EDF-Schedulability Problem**

**Input:** task system $\tau$

**Question:** is $\tau$ EDF-schedulable on one processor?

Let $U = \sum_i c_i/p_i$, $\varepsilon > 0$
### EDF-Schedulability Problem

**Input:** task system $\tau$

**Question:** is $\tau$ EDF-schedulable on one processor?

Let $U = \sum_i c_i / p_i$, $\varepsilon > 0$

Baruah, Rosier & Howell (1990) and subsequent work

EDF-Schedulability can be solved in pseudopolynomial time if $U < 1 - \varepsilon$. 
Schedulability Testing: EDF

**EDF-Schedulability Problem**

**Input:** task system $\tau$

**Question:** is $\tau$ EDF-schedulable on one processor?

Let $U = \sum_i c_i/p_i$, $\varepsilon > 0$

**Baruah, Rosier & Howell (1990) and subsequent work**

EDF-Schedulability can be solved in pseudopolynomial time if $U < 1 - \varepsilon$.

**Eisenbrand & Rothvoss (2010)**

EDF-Schedulability is (weakly) coNP-hard.
Zhang & Burns (2009): [...] the significant effort required to perform the exact schedulability test restricts the use of EDF in realistic systems.
Zhang & Burns (2009): [...] the significant effort required to perform the exact schedulability test restricts the use of EDF in realistic systems.

Unfortunately, the NP-hardness results make it unlikely that the efficiency of the generic tests can be improved.
Circumventing NP-Hardness

Zhang & Burns (2009): [...] the significant effort required to perform the exact schedulability test restricts the use of EDF in realistic systems.

Unfortunately, the NP-hardness results make it unlikely that the efficiency of the generic tests can be improved.

Idea: task sets occurring in some applications are structured, e.g. they have harmonic periods: $p_i | p_j$ or $p_j | p_i$ for all $i, j$
Circumventing NP-Hardness

Zhang & Burns (2009): [...] the significant effort required to perform the exact schedulability test restricts the use of EDF in realistic systems.

Unfortunately, the NP-hardness results make it unlikely that the efficiency of the generic tests can be improved.

Idea: task sets occurring in some applications are structured, e.g. they have harmonic periods: \( p_i \mid p_j \) or \( p_j \mid p_i \) for all \( i, j \).

Our Results

For task sets with harmonic periods and constrained deadlines:

- **Response Time Analysis** can be solved in polynomial time.
- **EDF-Schedulability** can be solved in polynomial time.
Related Work on Harmonic Tasksets

Our Results

For task sets with harmonic periods and constrained deadlines:

- **R**esponse **T**ime **A**nalysis can be solved in *polynomial* time
- **EDF-S**chedulability can be solved in *polynomial* time.
Related Work on Harmonic Tasksets

Our Results
For task sets with harmonic periods and constrained deadlines:

- **Response Time Analysis** can be solved in polynomial time
- **EDF-Schedulability** can be solved in polynomial time.

Earlier evidence that harmonic task sets are “better schedulable”:
Kuo & Mok (1991), Han & Tyan (1997), Chen, Mok & Kuo (2003)....
Our Results

For task sets with harmonic periods and constrained deadlines:

- **Response Time Analysis** can be solved in polynomial time.
- **EDF-Schedulability** can be solved in polynomial time.

Earlier evidence that harmonic task sets are “better schedulable”:
Kuo & Mok (1991), Han & Tyan (1997), Chen, Mok & Kuo (2003)...

Lehoczky, Sha & Ding (1989); Earlier folklore?

Rate Monotonic correctly schedules any harmonic task set with implicit deadlines and \( U \leq 1 \)

However, we consider constrained deadlines and arbitrary priorities.
FP-Schedulability
(Response Time Analysis)
Recall that the response time $r_n$ of $\tau_n$ is the least $t$ such that:

$$c_n + \sum_{i < n} \left\lceil \frac{t}{p_i} \right\rceil \cdot c_i \leq t. \quad (1)$$

Let $P = \max_i p_i$. Certainly $r_n \in [0, P \cdot c_n]$.

Idea: use binary search to look for $r_n$.

Let $p_j, p_i$ be “consecutive” periods (so $p_j \mid p_i$).

Say $r_n \in (\left( k - 1 \right) p_i, kp_i]$.

Condition (1) false – Condition (1) true!
Recall that the response time $r_n$ of $\tau_n$ is the least $t$ such that:

$$c_n + \sum_{i < n} \left\lceil \frac{t}{p_i} \right\rceil \cdot c_i \leq t. \quad (1)$$

Let $P = \max_i p_i$. Certainly $r_n \in [0, P \cdot c_n]$. 

**Response Time Analysis: A Binary Search Approach**
Recall that the response time $r_n$ of $\tau_n$ is the least $t$ such that:

$$c_n + \sum_{i < n} \left\lceil \frac{t}{p_i} \right\rceil \cdot c_i \leq t.$$  \hspace{1cm} (1)

Let $P = \max_i p_i$. Certainly $r_n \in [0, P \cdot c_n]$.

**Idea:** use binary search to look for $r_n$.
Response Time Analysis: A Binary Search Approach

Recall that the response time $r_n$ of $\tau_n$ is the least $t$ such that:

$$c_n + \sum_{i < n} \left\lfloor \frac{t}{p_i} \right\rfloor \cdot c_i \leq t. \quad (1)$$

Let $P = \max_i p_i$. Certainly $r_n \in [0, P \cdot c_n]$.

**Idea**: use binary search to look for $r_n$

Let $p_j, p_i$ be “consecutive” periods (so $p_j | p_i$)

Say $r_n \in ((k - 1)p_i, kp_i]$
Recall that the response time $r_n$ of $\tau_n$ is the least $t$ such that:

$$c_n + \sum_{i < n} \left\lceil \frac{t}{p_i} \right\rceil \cdot c_i \leq t. \quad (1)$$

Let $P = \max_i p_i$. Certainly $r_n \in [0, P \cdot c_n]$.

**Idea**: use binary search to look for $r_n$

Let $p_j, p_i$ be “consecutive” periods (so $p_j \mid p_i$)

Say $r_n \in ((k - 1)p_i, kp_i]$}

Condition (1) false – Condition (1) true!
Response Time Analysis: A Binary Search Approach

\[(k - 1)p_i \quad \? \quad \? \quad \? \quad \? \quad k \cdot p_i\]

\[(a - 1)p_j \quad a \cdot p_j\]

Condition (1) false – Condition (1) true!

Key Lemma

If periods are harmonic, then \( t(a) := a \cdot p_j \) satisfies Condition (1) for any \( a \in \mathbb{N} \) such that \( t(a) \in [r_n, k \cdot p_i] \).

\( \Rightarrow \) recurse on \( ((a - 1)p_j, a \cdot p_j] \)
Response Time Analysis: Algorithm and Summary

Key Lemma

If periods are harmonic, then \( t(a) := a \cdot p_j \) satisfies Condition (1) for any \( a \in \mathbb{N} \) such that \( t(a) \in [r_n, k \cdot p_i] \).
Response Time Analysis: Algorithm and Summary

Key Lemma
If periods are harmonic, then \( t(a) := a \cdot p_j \) satisfies Condition (1) for any \( a \in \mathbb{N} \) such that \( t(a) \in [r_n, k \cdot p_i] \).

Algorithm 1:
\[
\begin{aligned}
LB & \leftarrow 0 \\
UB & \leftarrow P \cdot c_n \quad \text{[invariant: } LB < r_n \leq UB]\n\end{aligned}
\]

repeat \( n-1 \) times (for decreasing values of \( p_j \)):
\[
\begin{aligned}
\text{Binary search the least } a \text{ such that } a \cdot p_j \text{ satisfies Condition (1)} \\
LB & \leftarrow (a - 1) \cdot p_j \\
UB & \leftarrow a \cdot p_j
\end{aligned}
\]

return \( c_n + \sum_{i<n} \left\lceil \frac{UB}{p_i} \right\rceil c_i \)
**Algorithm 1**:

\[ LB \leftarrow 0 \]
\[ UB \leftarrow P \cdot c_n \quad \text{[invariant: } LB < r_n \leq UB\text{]} \]

repeat \( n - 1 \) times (for decreasing values of \( p_j \)):

- Binary search the least \( a \) such that \( a \cdot p_j \) satisfies Condition (1)
  \[ LB \leftarrow (a - 1) \cdot p_j \]
  \[ UB \leftarrow a \cdot p_j \]

return \( c_n + \sum_{i < n} \left \lfloor UB/p_i \right \rfloor c_i \)

**Theorem**

Algorithm 1 is correct. Its running time is \( O(n \log P) \).
EDF-Schedulability
EDF-Schedulability: Procrastination

We analyze a non-EDF, yet optimal schedule: “Procrast” ("procrastinating")

We’ll show Procrast-schedulability \( \equiv \) EDF-Schedulability

Let \( p_1 \leq p_2 \leq \ldots \leq p_n \). Procrast schedule example:

Similar to a “backwards” Rate Monotonic schedule

Key Lemma

1. Procrast-schedulability can be tested in polynomial time;
2. Procrast is optimal (!)
**EDF-Schedulability: Procrastination**

We analyze a non-EDF, yet optimal schedule: “Procrast” (“procrastinating”)

We’ll show Procrast-schedulability $\equiv$ EDF-Schedulability

Let $p_1 \leq p_2 \leq \ldots \leq p_n$. Procrast schedule example:

![Procrast Schedule Example](image)

Similar to a “backwards” Rate Monotonic schedule

**Key Lemma**

1. Procrast-schedulability can be tested in polynomial time;
2. Procrast is optimal (!)
EDF-Schedulability: Procrastination

We analyze a non-EDF, yet optimal schedule: “Procrast” (“procrastinating”)

We’ll show Procrast-schedulability $\equiv$ EDF-Schedulability

Let $p_1 \leq p_2 \leq \ldots \leq p_n$. Procrast schedule example:

Similar to a “backwards” Rate Monotonic schedule

Key Lemma

1. Procrast-schedulability can be tested in polynomial time;
2. Procrast is optimal (!)
Procrast-schedulability can be tested in polynomial time

Due to harmonicity, starting time of $\tau_j$ relative to $p_j$ is a constant
Call it start offset ($b_j$). The $b_j$’s compactly describe the schedule

Easy to compute $b_j$ efficiently if we have an efficient procedure for $\text{IDLE}_j(x) := \text{idle time in } [x, d_j)$ (after fixing $\tau_1, \ldots, \tau_{j-1}$)

$\text{IDLE}_j$ allows to compute $b_j$ efficiently via binary search
Computing $IDLE_j(x)$

$IDLE_j(x) :=$ idle time in $[x, d_j)$ (after fixing $\tau_1, \ldots, \tau_{j-1}$)

Compute largest $x' \leq x$ such that $x'$ is not beyond the start offset of smaller period tasks (can be done recursively)

- During $[x', x)$ the processor must be busy (by construction)
- During $[0, x')$ the total work $C$ is on jobs already completed at $x'$ $\Rightarrow$
  easy to compute $C$

$IDLE'_j[0, x) = x' - C$. 
Computing $\text{IDLE}_j(x)$

$\text{IDLE}_j'(x) := \text{idle time in } [0, x) \text{ (after fixing } \tau_1, \ldots, \tau_{j-1})$

Compute largest $x' \leq x$ such that $x'$ is not beyond the start offset of smaller period tasks (can be done recursively)

- During $[x', x)$ the processor must be busy (by construction)
- During $[0, x')$ the total work $C$ is on jobs already completed at $x' \Rightarrow$ easy to compute $C$

$\text{IDLE}_j'(0, x) = x' - C$. 
Computing $IDLE_j(x)$

$IDLE'_j(x) :=$ idle time in $[0,x)$ (after fixing $\tau_1, \ldots, \tau_{j-1}$)

Compute largest $x' \leq x$ such that $x'$ is not beyond the start offset of smaller period tasks (can be done recursively)

- During $[x',x)$ the processor must be busy (by construction)
- During $[0,x')$ the total work $C$ is on jobs already completed at $x' \Rightarrow$ easy to compute $C$

$IDLE'_j[0,x) = x' - C.$
(2) Procrast is optimal

Lemma

For each $j$ and $x \geq 0$, the Procrast schedule for $\tau_1, \ldots, \tau_j$ maximizes the amount of idle time in $[0, x)$ (namely, no feasible schedule has more idle time).

Proof by induction. $j = 1$ holds by construction.
Lemma

For each $j$ and $x \geq 0$, the Procrast schedule for $\tau_1, \ldots, \tau_j$ maximizes the amount of idle time in $[0, x)$ (namely, no feasible schedule has more idle time).

Proof by induction. $j = 1$ holds by construction.

If Procrast construction fails

$\Rightarrow$ no schedule has $c_j$ idle time within $[0, d_j)$

$\Rightarrow$ taskset is infeasible.
Algorithm 2:
Assume $p_1 \leq p_2 \leq \ldots \leq p_n$ (if not, reorder the tasks)
for $j = 1$ to $n$ do:
    \[ \text{idlemax} \leftarrow \text{IDLE}_j(0) \]
    if idlemax $< c_j$ then return infeasible
    else
        \[ b_j \leftarrow \max\{x : \text{IDLE}_j(x) = c_j\} \]
return (feasible, $(b_1, b_2, \ldots, b_n)$)
Algorithm 2:
Assume $p_1 \leq p_2 \leq \ldots \leq p_n$ (if not, reorder the tasks)
for $j = 1$ to $n$ do:
    $\text{idlemax} \leftarrow \text{IDLE}_j(0)$
    if $\text{idlemax} < c_j$ then return infeasible
    else
        $b_j \leftarrow \max\{x : \text{IDLE}_j(x) = c_j\}$
    return (feasible, $(b_1, b_2, \ldots, b_n)$)

Theorem
Algorithm 2 is correct. Its running time is $O(n^3 \log P)$. 
A taskset is jointly harmonic if for all $x_i, x_j \in \{d_1, \ldots, d_n, p_1, \ldots, p_n\}$, either $x_i \mid x_j$ or $x_j \mid x_i$. 
A taskset is **jointly harmonic** if for all \( x_i, x_j \in \{ d_1, \ldots, d_n, p_1, \ldots, p_n \} \), either \( x_i \mid x_j \) or \( x_j \mid x_i \).

We give a faster/simpler test for jointly harmonic tasksets:

**Theorem**

There is a \( O(n^2) \) algorithm for EDF-schedulability of jointly harmonic tasksets.
A taskset is **jointly harmonic** if for all \( x_i, x_j \in \{d_1, \ldots, d_n, p_1, \ldots, p_n\} \), either \( x_i \mid x_j \) or \( x_j \mid x_i \).

We give a faster/simpler test for jointly harmonic tasksets:

**Theorem**

There is a \( O(n^2) \) algorithm for EDF-schedulability of jointly harmonic tasksets.

**Idea**: there is no **strictly crossing pair** of jobs in the synchronous arrival sequence.

A pair of jobs is **strictly crossing** if their scheduling windows overlap, without one being contained in the other.
A taskset is **jointly harmonic** if for all $x_i, x_j \in \{d_1, \ldots, d_n, p_1, \ldots, p_n\}$, either $x_i \mid x_j$ or $x_j \mid x_i$.

We give a faster/simpler test for jointly harmonic tasksets:

**Theorem**

There is a $O(n^2)$ algorithm for EDF-schedulability of jointly harmonic tasksets.

**Idea**: there is no strictly crossing pair of jobs in the synchronous arrival sequence

A pair of jobs is **strictly crossing** if their scheduling windows overlap, without one being contained in the other

The absence of strictly crossing pairs simplifies the schedule considerably
Conclusions and Further Directions

Harmonicity enables faster, provably efficient (polynomial) exact schedulability tests

- **Response Time Analysis** can be solved in time $O(n \log P)$
- **EDF-Schedulability** can be solved in time $O(n^3 \log P)$
  - improves to $O(n^2)$ if taskset is jointly harmonic
Harmonicity enables faster, provably efficient (polynomial) exact schedulability tests

- **Response Time Analysis** can be solved in time $O(n \log P)$
- **EDF-Schedulability** can be solved in time $O(n^3 \log P)$
  - improves to $O(n^2)$ if taskset is jointly harmonic

Open directions:

- extension to the arbitrary deadline case ($d_i \leq p_i$)
- extension to multiprocessors – are the problems NP-hard or not?
- improve the $O(n^3 \log P)$ bound to $O(n \log P)$?