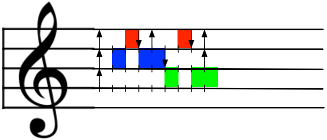
Polynomial-Time Exact Schedulability Tests for Harmonic Real-Time Tasks

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

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



Outline

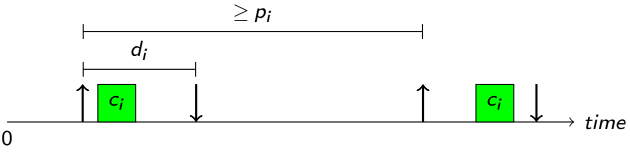
Sporadic Task Model – Constrained Deadlines

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

Task τ_i has:

- a worst-case computation time c_i
- a relative deadline d_i
- a minimum inter-arrival time (period) p_i

Example:



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

Schedulability Testing

A task system is ALG-schedulable if, for every of its (legal) job sequences, algorithm ALG constructs a feasible schedule

Schedulability Testing

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

Schedulability Testing

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

Remark. Enough to analyze the Synchronous Arrival Sequence:

- jobs of τ_i released at 0, $1 \cdot p_i$, $2 \cdot p_i$, $3 \cdot p_i$, $\ldots$

Schedulability Testing: FP

FP-schedulability is often verified via Response Time Analysis.

Schedulability Testing: FP

FP-schedulability is often verified via Response Time Analysis.

RESPONSE TIME ANALYSIS Problem

Input: priority-ordered task system τ, integer R
Question: is the response time of τ_n at most R?

Schedulability Testing: FP

FP-schedulability is often verified via Response Time Analysis.

Response Time Analysis Problem

Input: priority-ordered task system τ, integer R
Question: is the response time of τ_n at most R?

Liu & Layland (1973) and subsequent work

Response Time Analysis can be solved in pseudopolynomial time.

Schedulability Testing: FP

FP-schedulability is often verified via Response Time Analysis.

Response Time Analysis Problem

Input: priority-ordered task system τ, integer R
Question: is the response time of τ_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 τ
Question: is τ EDF-schedulable on one processor?

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

Schedulability Testing: EDF

EDF-SCHEDULABILITY Problem

Input: task system τ
Question: is τ 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 τ
Question: is τ 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.

Circumventing NP-Hardness

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

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

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

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:

- 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:

- 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)...

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)...

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)

Response Time Analysis: A Binary Search Approach

Recall that the response time r_n of τ_n is the least t such that:

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

Response Time Analysis: A Binary Search Approach

Recall that the response time r_n of τ_n is the least t such that:

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

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 τ_n is the least t such that:

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

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

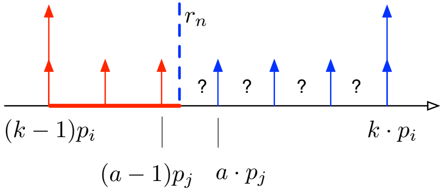
Idea: use binary search to look for r_n

Response Time Analysis: A Binary Search Approach

Recall that the response time r_n of τ_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 $r_n \in ((k-1)p_i, kp_i]$

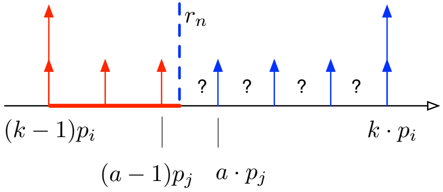


Response Time Analysis: A Binary Search Approach

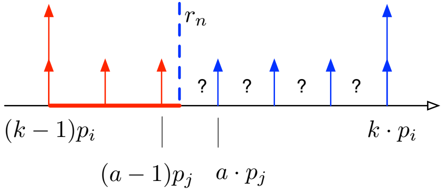
Recall that the response time r_n of τ_n is the least t such that:

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

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!



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 $t(a) \in [r_n, k \cdot p_i]$.

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:

$LB \leftarrow 0$

$UB \leftarrow P \cdot c_n$ *[invariant: $LB < r_n \leq UB$]*

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 $c_n + \sum_{i<n} \lceil UB/p_i \rceil c_i$

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:
$LB \leftarrow 0$
$UB \leftarrow P \cdot c_n$ *[invariant: $LB < r_n \leq UB$]*
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 $c_n + \sum_{i<n} \lceil UB/p_i \rceil c_i$

Theorem

Algorithm 1 is correct. Its running time is $O(n \log P)$.

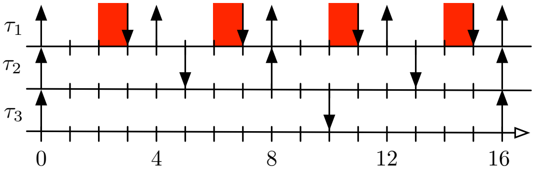
EDF-Schedulability

EDF-Schedulability: Procrastination

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

We'll show Procrast-schedulability ≡ 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

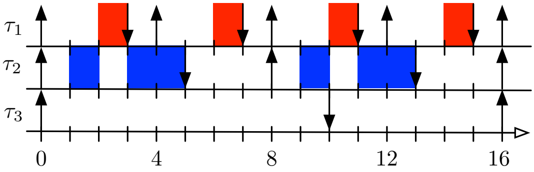
1. Procrast-schedulability can be tested in polynomial time;
2. Procrast is optimal (!)

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:



Similar to a "backwards" Rate Monotonic schedule

Key Lemma

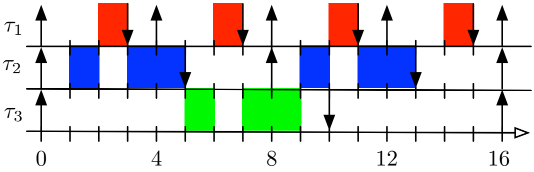
1. Procrast-schedulability can be tested in polynomial time;
2. Procrast is optimal (!)

EDF-SCHEDULABILITY: Procrastination

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

We'll show Procrast-schedulability ≡ EDF-Schedulability

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



Similar to a "backwards" Rate Monotonic schedule

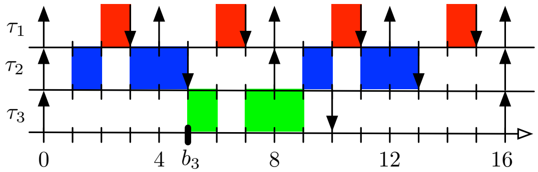
Key Lemma

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

(1) Procrast-schedulability can be tested in polynomial time

Due to harmonicity, starting time of τ_j relative to p_j is a constant

Call it start offset (b_j). The b_j's compactly describe the schedule



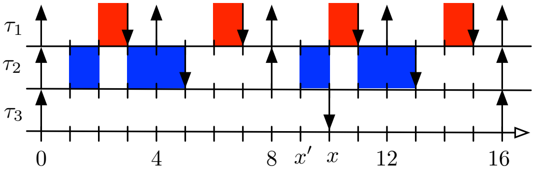
Easy to compute b_j efficiently if we have an efficient procedure for $IDLE_j(x) :=$ idle time in $[x, d_j)$ (after fixing $\tau_1, \ldots, \tau_{j-1}$)

$IDLE_j$ 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)



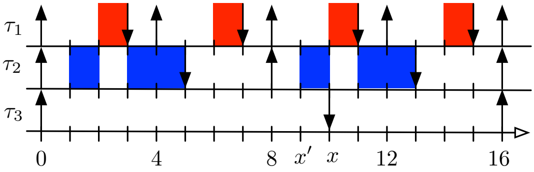
- During $[x', x)$ the processor must be busy (by construction)
- During $[0, x')$ the total work C is on jobs already completed at x' $\Rightarrow$ easy to compute C

$IDLE'_j[0, x) = x' - C$.

Computing $IDLE_j(x)$

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

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



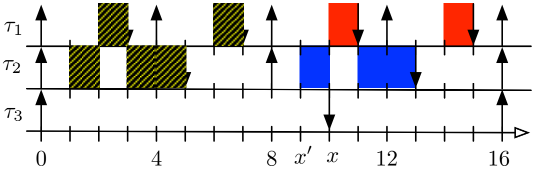
- During $[x', x)$ the processor must be busy (by construction)
- During $[0, x')$ the total work C is on jobs already completed at x' $\Rightarrow$ easy to compute C

$IDLE'_j[0,x) = x' - C.$

Computing $IDLE_j(x)$

$IDLE'_j(x) :=$ 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. □

(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. □

If Procrast construction fails
$\Rightarrow$ no schedule has c_j idle time within $[0, d_j)$
$\Rightarrow$ taskset is infeasible.

EDF-Schedulability: Algorithm and Summary

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

EDF-Schedulability: Algorithm and Summary

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

Theorem

Algorithm 2 is correct. Its running time is $O(n^3 \log P)$.

EDF-SCHEDULABILITY for "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 $x_j \mid x_i$.

EDF-SCHEDULABILITY for "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 $x_i \,|\, x_j$ or $x_j \,|\, x_i$.

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.

EDF-Schedulability for "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

EDF-Schedulability for "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 $x_i \,|\, x_j$ or $x_j \,|\, x_i$.

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

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

Open directions:

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