CSCE 990: *Real-Time Systems*

Intractability

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/RealTimeSystems

Intractability Results

- We now look at some intractability results regarding the "**feasibility problem**."
  - **Our Main Goal:** To develop some intuition about when a problem is likely to have a polynomial-time solution, a pseudo-polynomial-time solution, or probably neither.
  - **Outline:**
    - Preemptive, dynamic-priority systems: Baruah, Howell, & Rosier, 1993.
    - Preemptive, static-priority systems: Leung & Whitehead, 1982.
    - Nonpreemptive, dynamic-priority systems: Jeffay, Stanat, & Martel, 1991.
    - Nonpreemptive, static-priority systems: No reference for this.

The Feasibility Problem

- This is simply the problem of determining schedulability in an “if and only if” sense.
- We will consider both periodic and sporadic task systems.
  - » Tasks are still assumed to be independent.
- **Important Note:** In this part of the course, you should interpret “periodic” and “sporadic” to mean what the rest of the world thinks they mean.

Preemptive, Dynamic-Priority Systems

(Baruah, Howell, & Rosier)

- In this paper, relative deadlines may differ from periods.
- We will cover the following results from this paper:
  - A general demand-based technique for checking feasibility that always works, but requires further refinement to be practical.
  - A proof that the feasibility problem for asynchronous, periodic task systems is co-NP-complete in the strong sense.
  - An algorithm for determining feasibility for synchronous-periodic and sporadic task systems (which are equivalent **(why?)**) that runs in pseudo-polynomial time provided we cap utilization to be less than 1 (say U = 0.999). (!!)
    - This is a very useful trick that works for lots of “hard” problems.
    - It is not know whether the feasibility problem for such task systems is co-NP-hard in the strong sense.

Task Systems

Definition: A **task system** is a set $\tau = \{T_1, \ldots, T_n\}$ of tasks.

We assume tasks are independent and preemptive, and that time is discrete.

Definition: In a **periodic task system**, each task T_i is characterized by a 4-tuple (s_i, e_i, d_i, p_i) where:

s_i is its **start time** (i.e., release time);
$e_i \geq 1$ is its **execution time**;
$d_i \geq e_i$ is its **deadline**; and
$p_i \geq e_i$ is its **period**.

The meanings of these things should be obvious to you by now, so we won't bother to define what they mean.

Note: We can have $d_i \geq p_i$, $d_i = p_i$, or $d_i \leq p_i$.

Task Systems (Continued)

Definition: In a **sporadic task system**, each task T_i is characterized by a 3-tuple (e_i, d_i, p_i) where:

$e_i \geq 1$ is its **execution time**;
$d_i \geq e_i$ is its **deadline**; and
$p_i \geq e_i$ is its **minimum separation time**.

Again, the meanings of these things should be obvious to you by now, so we won't bother to define what they mean.

Note: Once again, we can have $d_i \geq p_i$, $d_i = p_i$, or $d_i \leq p_i$.

What Do We Know So Far?

We have shown that EDF is optimal for any type of task system that falls out of these definitions (e.g., synchronous/periodic with relative deadlines equal to periods, asynchronous/periodic with arbitrary deadlines, sporadic with arbitrary deadlines, etc.). [See Theorem 4-1.]

For any type of task system that falls out of these definitions such that each task's relative deadline is at least its period, feasibility can be determined by checking "Utilization ≤ 1". [See Theorem 6-1.]

Thus, any difficulties in determining feasibility are entirely due to the fact that relative deadlines can be less than periods.

- It is precisely this case where we had to turn to a "just-sufficient" condition based on densities. [See Theorem 6-2.]
- Unfortunately, tasks with relative deadlines < periods are common in practice.

A Feasibility Condition

Note: In this paper, σ is the EDF scheduling algorithm .

Also, $\underbrace{g_R(t_1, t_2)}_{\text{this is just demand that must be fulfilled in } [t_1, t_2]} = \Sigma_{(i,t) \in R(t_1,t_2)} e_i$, where $R(t_1, t_2) = \{(\overset{\text{task index}}{i},\overset{\text{release time}}{t}) \mid t_1 \leq t \leq t_2 - d_i\}$.

Theorem 2.2: Let $\tau = \{T_1, \ldots, T_n\}$ be a task system. Suppose τ is not feasible, and let R be a legal set of τ-requests for which σ reports failure at the earliest time t_f. Then for some $t_b < t_f$, $g_R(t_b, t_f) > t_f - t_b$.

Generalizes various demand-based arguments we've seen before.

Corollary

Corollary 2.3: Let $\tau = \{T_1, \ldots, T_n\}$ be a task system. τ is not feasible if and only if there exist a legal set of τ-requests R and natural numbers $t_1 < t_2$ such that $g_R(t_1, t_2) > t_1 - t_2$.

Two issues in applying this condition:

- To determine feasibility we have to consider <u>all</u> intervals $[t_1, t_2]$, which is obviously impossible (if the scheduled repeats at the LCM it's possible, but highly inefficient). For a particular type of task system, we would like to show that only a small **testing set** of time instants need be considered.

  Note: This is essentially what a "critical instant" argument does.

- We need to be able to compute $g_R(t_1, t_2)$ efficiently.

Proof of Theorem 2.2

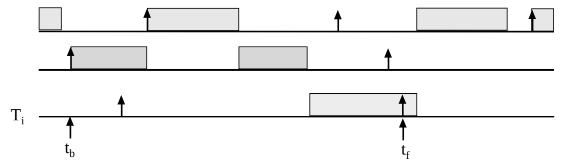
Suppose τ is not feasible. Let R_0 be a legal set of τ-requests such that σ reports failure for the first time at t_f.

Let $R_1 = R_0$ – all requests with deadlines after t_f.

With R_1, σ reports failure for the first time at t_f too. (Why?)

Let t_b be the latest instant such that the processor is idle in $[t_b - 1, t_b]$ (or 0, if the processor has never been idle).

Let $R_2 = R_1$ – all requests that execute in $[0,t_b)$.

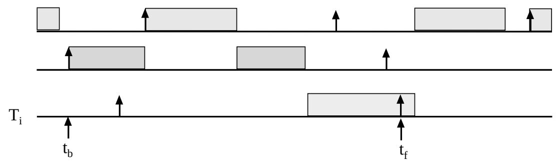


Proof of Theorem 2.2 (Continued)

With σ executing only the requests in R_2, we have the following properties:

- Each request in R_2 is released at or after t_b and has a deadline at or before t_f.
- The processor is continually busy executing requests in R_2 over the interval $[t_b, t_f)$.
- σ reports failure exactly once, at t_f.

It follows that $g_{R2}(t_b, t_f) > t_f - t_b$.



Periodic Task Systems

We will show that the feasibility problem is co-NP-complete in the strong sense for periodic task systems, and try to develop some intuitive understanding of why this is so.

First, we will prove a result that allows us to efficiently compute $g_{R_\tau}(t_1, t_2)$ for periodic task systems.

Lemma 3.1: Let $\tau = \{T_1, \ldots, T_n\}$ be a periodic task system. Then

$$g_{R_\tau}(t_1, t_2) = \sum_{i=1}^{n} e_i \cdot \max\left\{0, \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \max\left\{0, \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil\right\} + 1\right\}.$$

Proof of Lemma 3.1

$g_{R_\tau}(t_1,t_2) = \sum_{i=1}^{n} e_i \cdot c_{R_\tau}(i,t_1,t_2)$, where $c_{R_\tau}(i,t_1,t_2)$ is the number of requests in R_τ released at or after t_1 with deadlines at or before t_2.

$c_{R_\tau}(i,t_1,t_2)$ is the total number of natural numbers k satisfying both (1) and (2) below
(1) $t_1 \le s_i + k \cdot p_i$
(2) $s_i + k \cdot p_i + d_i \le t_2$

By (1), $k \ge (t_1 - s_i)/p_i$. The smallest natural number satisfying this is $\max\left\{0, \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil\right\}$.

Proof of Lemma 3.1 (Continued)

By (2), $k \le (t_2 - s_i - d_i)/p_i$. The largest integer satisfying this is $\left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor$.

Therefore, $c_{R_\tau}(i,t_1,t_2) = \max\left\{0, \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \max\left\{0, \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil\right\} + 1\right\}$.

Thus, $g_{R_\tau}(t_1,t_2) = \sum_{i=1}^{n} e_i \cdot \max\left\{0, \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \max\left\{0, \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil\right\} + 1\right\}$.

Towards a Reasonable "Testing Set"

The proof that the feasibility problem is in co-NP is based on the following result. This result (or a similar one) was actually first proved by Leung and Merrill.

Corollary 3.4: Suppose τ is not feasible and $\sum_{i=1,\ldots,n} e_i/p_i \leq 1$. Then there exist t_1 and t_2, where

$$0 \leq t_1 < t_2 \leq 2 \cdot P + \max_{1 \leq i \leq n} \{d_i\} + \max_{1 \leq i \leq n} \{s_i\}$$

such that $g_{R_\tau}(t_1, t_2) > t_2 - t_1$. ($P \equiv$ the LCM of the task periods.)

We will skip the proof of this result, but it is a very famous and important result, so you should know about it.

Informally, it says all "transient start-up effects" should die out by time $P + \max_{1 \leq i \leq n} \{d_i\} + \max_{1 \leq i \leq n} \{s_i\}$. Thus, if a deadline is missed *anywhere*, then one is missed by time $2 \cdot P + \max_{1 \leq i \leq n} \{d_i\} + \max_{1 \leq i \leq n} \{s_i\}$. This gives us a finite "testing set."

Theorem 3.5

Theorem 3.5: The feasibility problem for periodic task systems is in co-NP.

Proof:

We have to show that the "not feasible" problem is in NP.

Here's an algorithm: First, check whether $\sum_{i=1,\ldots,n} e_i/p_i \leq 1$. If so, nondeterministically guess t_1 and t_2 subject to the constraints of Corollary 3.4. Calculate $g_{R_\tau}(t_1, t_2)$ as in Lemma 3.1. If $g_{R_\tau}(t_1, t_2) > t_2 - t_1$, then answer "yes".

Simultaneous Congruences Problem

To prove that the feasibility problem for periodic task systems in co-NP-hard in the strong sense, we will transform from the **Simultaneous Congruences Problem (SCP)**.

SCP: Let $A = \{(a_1, b_1), \ldots, (a_n, b_n)\} \subseteq \mathbf{N} \times \mathbf{N}^+$ and $2 \le k \le n$ be given. The problem is to determine whether there is a subset $A' \subseteq A$ of k pairs and a natural number x such that, for every $(a_i, b_i) \in A'$, $x \equiv a_i \pmod{b_i}$.

Lemma 3.7: The SCP is NP-complete in the strong sense.

We will take this lemma on faith …

Strong co-NP-hardness

Theorem 3.8: [Leung and Merrill + Baruah et al.] The feasibility problem for periodic task systems is co-NP-hard in the strong sense.

Proof:

Let $\{(a_1, b_1), \ldots, (a_n, b_n)\}$ and k be an instance to the SCP. We will transform this instance to an equivalent instance of the "not feasible" problem.

For $1 \le i \le n$, let
$s_i = (k - 1)a_i$,
$e_i = 1$,
$d_i = k - 1$, and
$p_i = (k - 1)b_i$

Proof of Theorem 3.8 (Continued)

- **Observation 1:** Each task releases its first job at a time instant that is a multiple of k – 1, and its period is a multiple of k – 1; hence, *all* job releases occur at time instants that are multiples of k – 1.
- **Observation 2:** Because all tasks have a relative deadline of k – 1, if k (or more) tasks simultaneously release jobs, then a deadline will be missed.
- **Observation 3:** If a deadline is missed, then by Observation 1, k (or more) tasks must have simultaneously released jobs.
- **Observation 4:** By the previous three observations, **the task set is feasible if and only if k (or more) tasks never simultaneously release jobs**.

Proof of Theorem 3.8 (Continued)

Claim: k tasks simultaneously release jobs if and only if SCP answers "yes".

Proof:

"If": If SCP answers "yes" then $x \equiv a_i \pmod{b_i}$ hold for k of the pairs. Without loss of generality, assume that

$$x \equiv a_1 \pmod{b_1}$$
$$x \equiv a_2 \pmod{b_2}$$
$$\vdots$$
$$x \equiv a_k \pmod{b_k}.$$

Proof of Theorem 3.8 (Continued)

This implies that

$x \cdot (k - 1) \equiv a_1 \cdot (k - 1) \pmod{b_1 \cdot (k - 1)}$
$x \cdot (k - 1) \equiv a_2 \cdot (k - 1) \pmod{b_2 \cdot (k - 1)}$
$\vdots$
$x \cdot (k - 1) \equiv a_k \cdot (k - 1) \pmod{b_k \cdot (k - 1)}$.

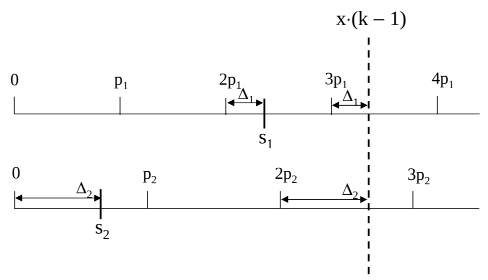
Or,

$x \cdot (k - 1) \equiv s_1 \pmod{p_1}$
$x \cdot (k - 1) \equiv s_2 \pmod{p_2}$
$\vdots$
$x \cdot (k - 1) \equiv s_k \pmod{p_k}$.

This implies that $T_1, \ldots, T_k$ all release jobs at time $x \cdot (k - 1)$.

Proof of Theorem 3.8 (Continued)

Here's a picture for k = 2.



Proof of Theorem 3.8 (Continued)

"Only if": By reasoning the "reverse" of the "if" proof, we can show that if k tasks releases jobs at some time $x \cdot (k - 1)$, then SCP answers "yes".

To conclude the proof, we note that the given transformation can be done in polynomial time.

Implications

- Theorem 3.8 tells us that the feasibility problem for periodic tasks has no pseudo-polynomial-time algorithm unless P = NP.
  - » **Note:** This result depends crucially on the fact that deadlines can be less than periods. (Convince yourself of this!).
- Of the scheduling conditions we've seen so far, the utilization-based ones take polynomial time to evaluate, and the demand-based ones take pseudo-polynomial time. (Convince yourself of this too!)

Capping Utilization Doesn't Help

Corollary 3.9: The feasibility problem for periodic task systems is co-NP-hard in the strong sense even if the systems are restricted to have processor utilization not greater than ε, where ε is any fixed positive constant.

Proof:

If we modify the construction of Theorem 3.8 by multiplying all the start times and periods by $c \geq n/(\varepsilon(k - 1)\cdot\min\{b_i\})$, then the proof still holds and utilization is at most ε.

The Punch Line

Theorem 3.10: The feasibility problem for periodic task systems is co-NP-complete in the strong sense even if the systems are restricted to have processor utilization not greater than ε, where ε is any fixed positive constant.

Sporadic Task Systems

- The authors formally show that, with regard to feasibility, sporadic systems and synchronous-periodic systems are the same.
  - » This result is pretty intuitive, so we will not bother to discuss the proof.
- The authors also show that the feasibility problem for sporadic systems is in co-NP.
  - » The question of whether this problem is co-NP-complete in the strong sense is left open.

A Very Useful Trick

Theorem 4.6: Let c be a fixed constant, $0 < c < 1$. The feasibility problem for synchronous systems is solvable in $O(n \cdot \max\{p_i - d_i\})$ time when utilization is at most c.

Proof:

Let $\tau = \{T_1, \ldots, T_n\}$ be a synchronous task system with utilization at most c. By Lemma 4.3, τ is not feasible if and only if, for some t_f, $g_{R_\tau}(0, t_f) > t_f$. We will show that t_f may be chosen to be less than

$$\frac{c}{1-c}\max\{p_i - d_i\}$$

Proof of Theorem 4.6 (Continued)

Suppose $g_{R_\tau}(0, t_f) > t_f$. Let $I = \{i \mid d_i \le t_f\}$.

By Lemma 3.1, we have

$$
\begin{aligned}
t_f &< g_{R_\tau}(0, t_f) \\
&= \sum_{i=1}^{n} e_i \cdot \max\left\{0, \left\lfloor \frac{t_f - d_i}{p_i} \right\rfloor + 1\right\} \\
&= \sum_{i \in I} \left(\left\lfloor \frac{t_f - d_i}{p_i} \right\rfloor + 1 \right) \cdot e_i \\
&\le \sum_{i \in I} \frac{t_f - d_i + p_i}{p_i} \cdot e_i \\
&= \sum_{i \in I} \left(\frac{t_f e_i}{p_i} + \frac{(p_i - d_i) e_i}{p_i} \right) \\
&\le c t_f + c \cdot \max\{p_i - d_i\}.
\end{aligned}
$$

Solving for t_f, we get

$$t_f < \frac{c}{1-c} \max\{p_i - d_i\}.$$

Proof of Theorem 4.6 (Continued)

Given this result, we have the following $O(n \cdot \max\{p_i - d_i\})$ algorithm.

```
t := 1;
tlim := [c/(c – 1)] · max{$p_i$ – $d_i$};
**while** t < tlim **and** t ≥ $\sum_{i=1}^{n} e_i \cdot \max\left\{0, \left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right\}$ **do**
        t := t + 1
**od**;
**if** t = tlim **then**
        **return** feasible
**else**
        **return** not feasible
```

Preemptive, Static-Priority Systems

(Leung and Whitehead)

- We know that for sporadic and synchronous-periodic systems, feasibility can be determined in pseudo-polynomial time.
  - Just use TDA or generalized TDA.
  - To the best of my knowledge, it is unknown whether feasibility can be checked for such systems in polynomial time.
- This paper shows that feasibility for asynchronous-periodic systems is co-NP-hard in the strong sense.
  - The paper also contains many results about multiprocessors, which we are skipping.

If the Priority Assignment is Given

Theorem 3.7: Given an asynchronous system R and a priority assignment ρ, the problem to decide whether or not the schedule produced by ρ is valid is co-NP-hard in the strong sense.

Note: In this and the next theorem, Leung and Whitehead say "NP-hard" when they should have said "co-NP-hard."

Also: The transformation in this and the next proof are from SCP, which at the time was only known to be NP-complete. Baruah et al. subsequently showed that SCP is NP-complete in the strong sense, so we can strengthen this and the next theorem to say "in the strong sense."

Proof of Theorem 3.7

Let $\{(a_1, b_1), \ldots, (a_n, b_n)\}$ and k be an instance to the SCP. We will transform this instance to an equivalent instance of the "not feasible" problem for n + 1 static-priority tasks, where *we get to choose the priority definition*.

For $1 \le i \le n$, let
$s_i = a_i$,
$e_i = 1/k$,
$d_i = b_i$, and
$p_i = b_i$.

This part is similar to the previous construction. Baruah et al. essentially scaled everything up by a factor of k to make everything integral (as required by their model).

For $i = n+1$, let
$s_{n+1} = 0$,
$e_{n+1} = 1/k$,
$d_{n+1} = 1$, and
$p_{n+1} = 1$.

Proof of Theorem 3.7 (Continued)

Finally, we let $\rho = \{T_1, \ldots, T_{n+1}\}$ (highest to lowest). The rest of the proof is similar to before.

First, note that T_{n+1}, which is of lowest priority, releases a job at every integral time instant.

Because every job of every task has an execution cost of 1/k, the system is schedulable if and only if it is never the case that k (or more) of the tasks $\{T_1, \ldots, T_n\}$ release jobs together simultaneously.

As before, $\{T_1, \ldots, T_n\}$ release jobs together if and only if SCP answers "yes."

So, the given task set is feasible if and only if SCP answers "no."

As before, the transformation takes polynomial time.

Observation

- In the previous proof, we had $d_i = p_i$ for each task T_i.
- Thus, the problem is co-NP-hard in the strong sense even if we restrict relative deadlines to be equal to periods.

If the Priority Assignment is *Not* Given

Theorem 3.8: Given an asynchronous system R, the problem to decide whether or not R is schedulable on one processor is co-NP-hard in the strong sense.

Proof:

Let $\{(a_1, b_1), \ldots, (a_n, b_n)\}$ and k be an instance to the SCP. We will transform this instance to an equivalent instance of the "not feasible" problem for n static-priority tasks. *In this case, the priority definition isn't determined by us as part of the transformation.*

For $1 \le i \le n$, let

$s_i = a_i$,

$e_i = 1/(k - 1)$,

$d_i = 1$, and

$p_i = b_i$.

Proof of Theorem 3.8 (Continued)

The proof is very similar to the previous two such proofs.

Because every job of every task has an execution cost of 1/(k – 1) and a relative deadline of 1, the system is schedulable if and only if it is never the case that k (or more) of the tasks $\{T_1, \ldots, T_n\}$ release jobs together simultaneously.

As before, $\{T_1, \ldots, T_n\}$ release jobs together if and only if SCP answers "yes."

So, the given task set is feasible if and only if SCP answers "no."

As before, the transformation takes polynomial time.

Why do we need two proofs (i.e., can't we just use the Theorem 3.8 proof for Theorem 3.7)?

Dynamic-Priority, Nonpreemptive Systems

(Jeffay, Stanat, and Martel)

- What do we know so far about these systems?
  - For nonpreemptive systems, the worst-case phasing occurs when a low-priority job is released "right before" other jobs.
    - This is different from preemptive systems, where the worst-case phasing is when all tasks release jobs together.
    - For this reason, Jeffay et al. use the terminology "**concrete**" rather than "synchronous" to talk about a task system where the initial releases are fixed.
  - We previously covered the first part of this paper, where it is shown that feasibility can be checked in pseudo-polynomial time for sporadic and non-concrete-periodic task systems.
    - It is assumed in this paper that relative deadlines equal periods.
    - The Baruah et al. proof can be applied to show that the feasibility problem for concrete, periodic tasks with relative deadlines at most periods is co-NP-complete in the strong sense.

Asynchronous vs. Non-concrete

(A Comment on Terminology)

- Instead of synchronous and asynchronous, we will use the terms concrete and non-concrete to talk about non-preemptive systems.
  - » The term "asynchronous" means that either the initial phasing is arbitrary, or it is specified, but all tasks don't release their first jobs together.
    - Until now, the difference between these two alternatives hasn't really been important.
  - » For a non-concrete task system to be feasible, the system must be schedulable for *any* initial phasing.

Concrete Periodic Tasks

Theorem 5-2: Non-preemptive scheduling of concrete periodic tasks (SCPT) is NP-hard in the strong sense.

Proof:

The transformation is based on 3-Partition, defined as follows.

3-Partition: We are given a finite set A of 3m elements, a bound $B \in \mathbf{Z}^+$, and a "size" $s(a) \in \mathbf{Z}^+$ for each $a \in A$, such that each s(a) satisfies $B/4 < s(a) < B/2$ and $\sum_{j=1,\ldots,3m} s(a_j) = Bm$.

The problem is to determine if A can be partitioned into m disjoint sets $S_1, S_2, \ldots, S_m$ such that, for $1 \le i \le m$, $\sum_{a \in S_i} s(a) = B$. (With the above constraints, every S_i will contain exactly three elements from A.)

Proof of Theorem 5-2 (Continued)

Let $A = \{a_1, a_2, \ldots, a_{3m}\}$, B, $s(a_1), s(a_2), \ldots, s(a_{3m})$ be an arbitrary instance of 3-Partition.

We create an instance to the SCPT problem as follows. We have $n = 3m + 2$ tasks, defined as follows.

For $i = 1$,	For $i = 2$,	For $3 \le i \le 3m + 2$,
$s_1 = 0$,	$s_2 = 9B$,	$s_i = 0$,
$e_1 = 8B$,	$e_2 = 23B$,	$e_i = s(a_{j-2})$,
$d_1 = 20B$, and	$d_2 = 40B$, and	$d_i = 40Bm$, and
$p_1 = 20B$.	$p_2 = 40B$.	$p_i = 40Bm$.

This task set clearly can be constructed in polynomial time.

Proof of Theorem 5-2 (Continued)

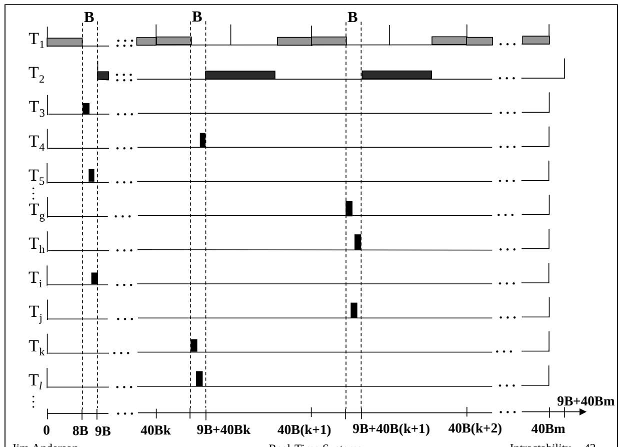
Note that total utilization is exactly one:

$$\sum_{j=1}^{n} \frac{c_j}{p_j} = \frac{8}{20} + \frac{23}{40} + \frac{\sum_{j=1}^{3m} s(a_j)}{40Bm} = \frac{39}{40} + \frac{Bm}{40Bm} = 1$$

Also, the system is feasible if and only if a schedule like the one on the following slide exists (this slide shows an EDF schedule; *any* correct schedule, EDF or not, must have this same general structure).

Note that, in this schedule, the jobs of tasks $T_3, \ldots, T_{3m+2}$ are partitioned into m groups of three, where the sum of the execution costs in each group is B.

Thus, the system is feasible iff the answer to 3-Partition is "yes".



Some Observations

- This proof doesn't appear to easily generalize to give us a nonpreemptive version of Leung and Whitehead's Theorem 3.7.
  - » We could prioritize T_1 over T_2, and T_2 over all other tasks, but we don't know how to prioritize T_3, …, T_{3m+2}.
- The Jeffay et al. proof does appear to generalize to give us a nonpreemptive version of Leung and Whitehead's Theorem 3.8, because in that proof, we don't have to specify a priority definition.
- **Feel free to try to prove either of these statements wrong!**

Static-Priority, Nonpreemptive Systems

- We can incorporate a blocking term in the TDA or generalized TDA approach to account for the amount of time a task can block on a nonpreemptive lower-priority job.
- The resulting schedulability check will be necessary and sufficient for sporadic or non-concrete periodic systems because we can adjust job releases to make the worst-case blockings actually happen.
- This test is only sufficient for concrete periodic task systems.

Static-Priority, Nonpreemptive(Cont'd)

- For nonpreemptive, concrete periodic task systems, Leung and Whitehead's proofs of Theorems 3.7 and 3.8 seem to work.
- So, we have a co-NP-complete in the strong sense result for:
  - » Nonpreemptive, concrete, periodic, specified priority definition, relative deadlines = periods (or ≤ or ≥). (Theorem 3.7.)
  - » Nonpreemptive, concrete, periodic, arbitrary priority definition, relative deadlines ≤ periods (or <). (Theorem 3.8.)
- As pointed out previously, the Jeffay et al. proof can be modified to get a NP-hard in the strong sense result for:
  - » Nonpreemptive, concrete, periodic, arbitrary priority definition, relative deadlines = periods.

Summary

- Let us categorize each task model by a 5-tuple A/B/C/D/E:
  - **A** is either **P** (preemptive) or **NP** (nonpreemptive).
  - **B** is either **DP** (dynamic priority), **SP**ρ (static priority definition ρ is given), or **SP** (static with arbitrary priority definition).
  - **C** is either **A or NC** (asynchronous or non-concrete) or **S or C** (synchronous or concrete). We use A and S for preemptive systems, and C and NC for nonpreemptive systems.
  - **D** is either **Per** (periodic) or **Spr** (sporadic).
  - **E** is either "**d=p**", "**d ≤ p**", or "**d?p**", indicating how deadlines relate to periods.
  - This gives us 72 task models!
  - The following slides summarize what's known about each.
  - **You are hereby challenged to find any errors in these tables!**

P/DP Systems (Preemptive, Dynamic Priorities)

Task Model	**Optimal Sched. Alg.**	**Feasibility in Poly. Time?**	**Feasibility in Pseudo. P.T.?**
P/DP/A/Per/d=p	EDF	Yes (can have d≥p)	Yes (can have d≥p)
P/DP/A/Per/d≤p	EDF	No (co-NPH-SS)	No (co-NPH-SS)
P/DP/A/Per/d?p	EDF	No (co-NPH-SS)	No (co-NPH-SS)
P/DP/A/Spr/d=p	EDF	Yes (can have d≥p)	Yes (can have d≥p)
P/DP/A/Spr/d≤p	EDF	Don't know	Don't know
P/DP/A/Spr/d?p	EDF	Don't know	Don't know
P/DP/S/Per/d=p	EDF	Yes (can have d≥p)	Yes (can have d≥p)
P/DP/S/Per/d≤p	EDF	Don't know	Don't know
P/DP/S/Per/d?p	EDF	Don't know	Don't know
P/DP/S/Spr/d=p	EDF	Yes (can have d≥p)	Yes (can have d≥p)
P/DP/S/Spr/d≤p	EDF	Don't know	Don't know
P/DP/S/Spr/d?p	EDF	Don't know	Don't know

P/SPρ Systems (Preemptive, Static ρ-Priorities)

Task Model	Optimal Sched. Alg.	Feasibility in Poly. Time?	Feasibility in Pseudo. P.T.?
P/SPρ/A/Per/d=p	N/A	No (co-NPH-SS)	No (co-NPH-SS)
P/SPρ/A/Per/d≤p	N/A	No (co-NPH-SS)	No (co-NPH-SS)
P/SPρ/A/Per/d?p	N/A	No (co-NPH-SS)	No (co-NPH-SS)
P/SPρ/A/Spr/d=p	N/A	Don't know	Yes (TDA)
P/SPρ/A/Spr/d≤p	N/A	Don't know	Yes (TDA)
P/SPρ/A/Spr/d?p	N/A	Don't know	Yes (Gen. TDA)
P/SPρ/S/Per/d=p	N/A	Don't know	Yes (TDA)
P/SPρ/S/Per/d≤p	N/A	Don't know	Yes (TDA)
P/SPρ/S/Per/d?p	N/A	Don't know	Yes (Gen. TDA)
P/SPρ/S/Spr/d=p	N/A	Don't know	Yes (TDA)
P/SPρ/S/Spr/d≤p	N/A	Don't know	Yes (TDA)
P/SPρ/S/Spr/d?p	N/A	Don't know	Yes (Gen. TDA)

P/SP Systems (Preemptive, Static Priorities)

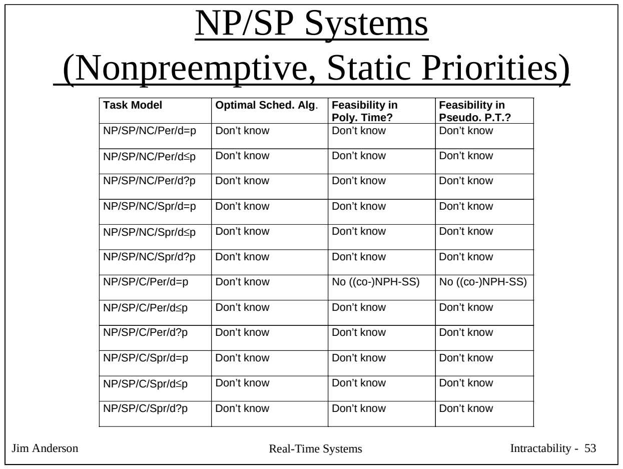
Task Model	Optimal Sched. Alg.	Feasibility in Poly. Time?	Feasibility in Pseudo. P.T.?
P/SP/A/Per/d=p	RM	Don't know	Don't know
P/SP/A/Per/d≤p	DM	No (co-NPH-SS)	No (co-NPH-SS)
P/SP/A/Per/d?p	Don't know	No (co-NPH-SS)	No (co-NPH-SS)
P/SP/A/Spr/d=p	RM	Don't know	Yes (TDA)
P/SP/A/Spr/d≤p	DM	Don't know	Yes (TDA)
P/SP/A/Spr/d?p	Don't know	Don't know	Yes (Gen. TDA)
P/SP/S/Per/d=p	RM	Don't know	Yes (TDA)
P/SP/S/Per/d≤p	DM	Don't know	Yes (TDA)
P/SP/S/Per/d?p	Don't know	Don't know	Yes (Gen. TDA)
P/SP/S/Spr/d=p	RM	Don't know	Yes (TDA)
P/SP/S/Spr/d≤p	DM	Don't know	Yes (TDA)
P/SP/S/Spr/d?p	Don't know	Don't know	Yes (Gen. TDA)

NP/DP Systems (Nonpreemptive, Dynamic Priorities)

Task Model	Optimal Sched. Alg.	Feasibility in Poly. Time?	Feasibility in Pseudo. P.T.?
NP/DP/NC/Per/d=p	EDF	Don't know	Yes
NP/DP/NC/Per/d≤p	Don't know	Don't know	Don't know
NP/DP/NC/Per/d?p	Don't know	Don't know	Don't know
NP/DP/NC/Spr/d=p	EDF	Don't know	Yes
NP/DP/NC/Spr/d≤p	Don't know	Don't know	Don't know
NP/DP/NC/Spr/d?p	Don't know	Don't know	Don't know
NP/DP/C/Per/d=p	Don't know	No (NHP-SS)	No (NHP-SS)
NP/DP/C/Per/d≤p	Don't know	No (co-NPH-SS)	No (co-NPH-SS)
NP/DP/C/Per/d?p	Don't know	No (co-NPH-SS)	No (co-NPH-SS)
NP/DP/C/Spr/d=p	EDF	Don't know	Yes
NP/DP/C/Spr/d≤p	Don't know	Don't know	Don't know
NP/DP/C/Spr/d?p	Don't know	Don't know	Don't know

NP/SPρ Systems (Nonpreemptive, Static ρ-Priorities)

Task Model	Optimal Sched. Alg.	Feasibility in Poly. Time?	Feasibility in Pseudo. P.T.?
NP/SPρ/NC/Per/d=p	N/A	Don't know	Yes (TDA w/ blocking term)
NP/SPρ/NC/Per/d≤p	N/A	Don't know	Yes (TDA w/ blocking term)
NP/SPρ/NC/Per/d?p	N/A	Don't know	Yes (Gen. TDA w/ blocking term)?
NP/SPρ/NC/Spr/d=p	N/A	Don't know	Yes (TDA w/ blocking term)
NP/SPρ/NC/Spr/d≤p	N/A	Don't know	Yes (TDA w/ blocking term)
NP/SPρ/NC/Spr/d?p	N/A	Don't know	Yes (Gen. TDA w/ blocking term)?
NP/SPρ/C/Per/d=p	N/A	No (co-NPH-SS)	No (co-NPH-SS)
NP/SPρ/C/Per/d≤p	N/A	No (co-NPH-SS)	No (co-NPH-SS)
NP/SPρ/C/Per/d?p	N/A	No (co-NPH-SS)	No (co-NPH-SS)
NP/SPρ/C/Spr/d=p	N/A	Don't know	Yes (TDA w/ blocking term)
NP/SPρ/C/Spr/d≤p	N/A	Don't know	Yes (TDA w/ blocking term)
NP/SPρ/C/Spr/d?p	N/A	Don't know	Yes (Gen. TDA w/ blocking term)?



NP/SP Systems (Nonpreemptive, Static Priorities)

Task Model	Optimal Sched. Alg.	Feasibility in Poly. Time?	Feasibility in Pseudo. P.T.?
NP/SP/NC/Per/d=p	Don’t know	Don’t know	Don’t know
NP/SP/NC/Per/d≤p	Don’t know	Don’t know	Don’t know
NP/SP/NC/Per/d?p	Don’t know	Don’t know	Don’t know
NP/SP/NC/Spr/d=p	Don’t know	Don’t know	Don’t know
NP/SP/NC/Spr/d≤p	Don’t know	Don’t know	Don’t know
NP/SP/NC/Spr/d?p	Don’t know	Don’t know	Don’t know
NP/SP/C/Per/d=p	Don’t know	No ((co-)NPH-SS)	No ((co-)NPH-SS)
NP/SP/C/Per/d≤p	Don’t know	Don’t know	Don’t know
NP/SP/C/Per/d?p	Don’t know	Don’t know	Don’t know
NP/SP/C/Spr/d=p	Don’t know	Don’t know	Don’t know
NP/SP/C/Spr/d≤p	Don’t know	Don’t know	Don’t know
NP/SP/C/Spr/d?p	Don’t know	Don’t know	Don’t know