A-periodic scheduling

- **Given:**
  - A set of non-periodic tasks \{J_1, \ldots, J_n\} with
    - arrival times \(a_i\), deadlines \(d_i\), computation times \(C_i\)
    - precedence constraints
    - resource constraints
  - Class of scheduling algorithm:
    - Preemptive, non-preemptive
    - Off-line / on-line
    - Optimal / heuristic
    - One processor / multi-processor
    - ...
  - Cost function:
    - Minimize maximum lateness (soft RT)
    - Minimize maximum number of late tasks (feasibility! – hard RT)
- **Find:**
  - Optimal / good schedule according to given cost function
Case 1: Aperiodic tasks with synchronous release

- A set of (a-periodic) tasks \( \{J_1, \ldots, J_n\} \) with
  - arrival times \( a_i = 0 \ \forall 1 \leq i \leq n \), i.e. “synchronous” arrival times
  - deadlines \( d_i \),
  - computation times \( C_i \)
  - no precedence constraints, no resource constraints, i.e. “independent tasks”

- non-preemptive
- single processor
- Optimal
- Find schedule which minimizes maximum lateness (variant: find feasible solution)
EDD – Earliest Due Date

EDD: execute the tasks in order of non-decreasing deadlines

- **Lemma:** If arrival times are synchronous, then preemption does not help, i.e. if there is a preemptive schedule with maximum lateness $L_{\text{max}}$, then there is also a non-preemptive schedule with maximum lateness $L_{\text{max}}$.

- **Theorem (Jackson ’55):** Given a set of $n$ independent tasks with synchronous arrival times, any algorithm that executes the tasks in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness.
Case 2: aperiodic tasks with asynchronous release

- A set of (a-periodic) tasks \( \{J_1, \ldots, J_n\} \) with
  - arbitrary arrival times \( a_i \)
  - deadlines \( d_i \)
  - computation times \( C_i \)
  - no precedence constraints, no resource constraints, i.e. "independent tasks"

- preemptive

- Single processor

- Optimal

- Find schedule which minimizes maximum lateness (variant: find feasible solution)
EDF – Earliest Deadline First

▪ **EDF**: At every instant execute the task with the earliest absolute deadline among all the ready tasks.

▪ **Theorem (Horn ’74)**:
  Given a set of $n$ independent task with arbitrary arrival times, any algorithm that at every instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.
Non-preemptive version

- **Changed problem:**
  - A set of (a-periodic) tasks \{J_1, \ldots, J_n\} with
    - **arbitrary** arrival times \(a_i\)
    - deadlines \(d_i\)
    - computation times \(C_i\)
    - no precedence constraints, no resource constraints, i.e. “independent tasks”
  - **Non-preemptive** instead of preemptive scheduling!
  - Single processor
  - Optimal
  - Find schedule which minimizes maximum lateness (variant: find feasible solution)
Non-preemptive version

- **Theorem** (Jeffay et al. ’91): EDF is an optimal *non-idle* scheduling algorithm also in a *non-preemptive* task model.

- When idle schedules are allowed: problem is NP-hard.
- **Possible approaches:**
  - Heuristics
  - Bratley’s algorithm: branch-and-bound
Case 3: Scheduling with precedence constraints

- Non-preemptive scheduling with non-synchronous arrival times, deadlines and precedence constraints is NP-hard.

Here:
- Restrictions:
  - Consider synchronous arrival times (all tasks arrive at 0)
  - Allow preemption.
- 2 different algorithms:
  - Latest deadline “first” (LDF)
  - Modified EDF

- Precedences define a partial order, represented as a DAG
- Scheduling determines a compatible total order
- Method: Topological sorting
Example

<table>
<thead>
<tr>
<th></th>
<th>J_1</th>
<th>J_2</th>
<th>J_3</th>
<th>J_4</th>
<th>J_5</th>
<th>J_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C_i</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d_i</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Diagram:

- J_1
- J_2
- J_3
- J_4
- J_5
- J_6
Example

- One of the following algorithms is optimal. Which one?

Algorithm 1:
1. Among all sources in the precedence graph select the task T with earliest deadline. Schedule T first.
2. Remove T from G.
3. Repeat.

Algorithm 2:
1. Among all sinks in the precedence graph select the task T with latest deadline. Schedule T last.
2. Remove T from G.
3. Repeat.

Forward topological sorting

Backward topological sorting
Example (continued)

- Algorithm 1:

<table>
<thead>
<tr>
<th></th>
<th>J_1</th>
<th>J_2</th>
<th>J_3</th>
<th>J_4</th>
<th>J_5</th>
<th>J_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C_i</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d_i</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

J_4 missed the deadline
Example (continued)

- Algorithm 2:

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
<th>$J_5$</th>
<th>$J_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$d_i$</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

$J_1$ $J_2$ $J_4$ $J_3$ $J_5$ $J_6$

d_1 d_4 d_3 d_5 d_2 d_6

$J_1$ $J_2$ $J_4$ $J_3$ $J_5$ $J_6$

Flexible
Example (continued)

- Algorithm 1 is **not** optimal.
- Algorithm 1 is the generalization of EDF to the case with precedence constraints.

- Is Algorithm 2 optimal?
- Algorithm 2 is called Latest Deadline First (LDF).

- **Theorem (Lawler 73):**
  LDF is optimal wrt. maximum lateness.
Proof of optimality

Task set \( J = \{J_1, \ldots, J_n\} \), \( 1 \leq n \) without successors.

Let \( J_e \in \mathcal{P} \) with the latest deadline.

Consider a schedule \( \pi \) satisfying constraints

where the last scheduled task is \( J_e \), \( k \neq l \), \( d_k < d_l \).

It is clear that \( J_e \in \mathcal{P} \).

We show that we can move \( J_e \) to the end of the schedule with

1) no violation of the precedence
2) no increase in max. lateness.
1. Precedence not violated: Je have no successe.

2. \( L'_\text{max} = \max \left\{ L'_\text{max}(A), L'_\text{max}(B), L'_{K}, L'_{e} \right\} \)

\( L'_\text{max}(A) = L'_\text{max}(A) \) nothing changed.

\( L'_\text{max}(B) < L'_\text{max}(B) \) starts and stops earlier.

\( L'_{K} < L'_{K} \) starts and ends earlier.

\( L'_{e} = \frac{n}{\sum_{i=1}^{n} c_{i}} = d_{e} < \frac{1}{2} \sum_{i=1}^{n} c_{i} - d_{e} \)

Remove \( J_{K} \) from \( J_{1} \) and continue.
Optimal scheduling algorithms for periodic tasks
Periodic scheduling

- **Given:**
  - A set of periodic tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with
    - phases $\Phi_i$ (arrival times of first instances of tasks),
    - periods $T_i$ (time difference between two consecutive activations)
    - relative deadlines $D_i$ (deadline relative to arrival times of instances)
    - computation times $C_i$
  
  $\Rightarrow j$th instance $\tau_{i,j}$ of task $\tau_i$ with
    - arrival time $a_{i,j} = \Phi_i + (j-1)T_i$,
    - deadline $d_{i,j} = \Phi_i + (j-1)T_i + D_i$

- **Find a feasible schedule**
  - start time $s_{i,j}$ and
  - finishing time $f_{i,j}$
Assumptions

A.1. Instances of periodic task $\tau_i$ are regularly activated with constant period $T_i$.
A.2. All instances have same worst case execution time $C_i$.
A.3. All instances have same relative deadline $D_i$, here in most cases equal to $T_i$ (i.e., $d_{i,j} = \Phi_i + j \cdot T_i$)
A.4. All tasks in $\Gamma$ are independent. No precedence relation, no resource constraints.
A.5. Overhead for context switches is neglected, i.e. assumed to be 0 in the theory.

- Basic results based on these assumptions form the core of scheduling theory.
- For practical applications, assumptions A.3. and A.4. can be relaxed, but results have to be extended.
Examples for periodic scheduling (1)

- Schedulable, but only preemptive schedule possible.
Examples for periodic scheduling (2)

- Schedulable with non-preemptive schedule.
Examples for periodic scheduling (3)

<table>
<thead>
<tr>
<th></th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Phi_i$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_i$</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$C_i$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$D_i$</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

$T_1 \cdot T_2 = 12$

Number of executions of $\tau_1$ and $\tau_2$ within 12 time units:

$\frac{12}{3} = 4$ executions of $\tau_1$

$\frac{12}{4} = 3$ executions of $\tau_2$

- No feasible schedule for single processor.

$4 \cdot 2 = 8$ time units by $\tau_1$

$3 \cdot 2 = 6$ time units by $\tau_2$

$\sum_i^\gamma T_i = \gamma \cdot T_i$

Number of activations of $\tau_i$ in $\gamma$ time

Taken by $\frac{1}{T_i}$

$\sum \frac{T_i}{C_i} \cdot C_i$ total time

$\sum \frac{T_i}{C_i}$
Processor utilization

Definition:
Given a set $\Gamma$ of $n$ periodic tasks, the \textbf{processor utilization} $U$ is given by

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}.$$
Processor utilization: using it as a schedulability criterion

- Given: a scheduling algorithm A
- Define $U_{\text{bnd}}(A) = \inf \{U(\Gamma) \mid \Gamma \text{ is not schedulable by algorithm } A\}$. 

If $U_{\text{bnd}}(A) > 0$ then a simple, sufficient criterion for schedulability by A can be based on processor utilization:

- If $U(\Gamma) < U_{\text{bnd}}(A)$ then $\Gamma$ is schedulable by A.
- However, if $U_{\text{bnd}}(A) < U(\Gamma) \leq 1$, then $\Gamma$ may or may not be schedulable by A.

- Question:
  Does a scheduling algorithm A exist with $U_{\text{bnd}}(A) = 1$?
Question:
Does a scheduling algorithm A exist with $U_{\text{bnd}}(A) = 1$?

Answer:
- No, if $D_i < T_i$ allowed.
- Example:

<table>
<thead>
<tr>
<th></th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Phi_i$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_i$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$D_i$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Yes, if $D_i = T_i$ (or $D_i \geq T_i$) → Earliest Deadline First (EDF)
- In the following: assume $D_i = T_i$. 

\[
\frac{1}{2} + \frac{1}{2} = 1
\]
Earliest Deadline First (EDF)

- EDF is applicable to both periodic and a-periodic tasks.

- If there are only periodic tasks, priority-based schemes like “rate monotonic scheduling (RM)” (see later) are often preferred, since
  - They are simpler due to fixed priorities ⇒ use in “standard OS” possible
  - sorting wrt. to deadlines at run time is not needed
EDF and processor utilization factor

- **Theorem:** A set of periodic tasks \( \tau_1, \ldots, \tau_n \) with \( D_i = T_i \) is schedulable with EDF iff \( U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \).

**Proof:**

Let \( \sum = T_1 \cdot T_2 \cdots T_n \)

\[
\frac{T_1}{T_i} \cdot C_i \text{ time taken by task } \tau_i \text{ within } \sum
\]

\[
= \sum \frac{C_i}{T_i} \cdot T = U \cdot T
\]

Assume \( U > 1 \). Then \( U \cdot T > T \)

The task set is not schedulable.
by definition: Assume task not schedulable, but has time overrun at \( t_2 \) in EDF schedule.

Let \([t_1, t_2]\) be the longest interval s.t.

- no idle times
- only instances with deadlines \( \leq t_2 \) executed.
- \( t_1 \) must be release time of an instance.

Time overflow at \( t_2 \):

\[
\Rightarrow (t_2 - t_n) < 2 \sum \xi_i = 2 \sum_{i=1}^{n} \frac{t_2 - t_n}{T_i} \cdot \xi_i \leq 2 \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} \cdot \xi_i
\]

\[
= (t_2 - t_n) \cdot \frac{1}{2} \sum_{i=1}^{n} \frac{\xi_i}{T_i} = (t_2 - t_n) \cdot \mu \Rightarrow \mu > \lambda \quad \forall
\]
Rate monotonic scheduling (RM) (Liu, Layland ’73):

- Assign **fixed priorities** to tasks $\tau_i$:
  - $\text{priority}(\tau_i) = 1/T_i$.
  - I.e., priority reflects release rate
- Always execute ready task with highest priority
- Preemptive: currently executing task is preempted by newly arrived task with shorter period.
Example for RM (1)

\[
\begin{array}{c|ccc}
\Phi_i & \tau_1 & \tau_2 & \tau_3 \\
\hline
0 & 0 & 0 & 0 \\
T_i & 4 & 6 & 12 \\
C_i & 2 & 1 & 4 \\
D_i & 4 & 6 & 12 \\
\end{array}
\]

\[
\min \sigma(\tau) = \frac{1}{4} > \frac{1}{6} \quad \min \sigma(\tau_2) = \frac{1}{6} > \frac{1}{12} \quad \min \sigma(\tau_3) = \frac{1}{12}
\]

Feasible schedule by RM
Example for RM (2)

<table>
<thead>
<tr>
<th>$\Phi_i$</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_i$</td>
<td>4</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>$C_i$</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$D_i$</td>
<td>4</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

$\mu_{\Phi}(\tau_1) = \frac{1}{4}$

$\mu_{\Phi}(i_2) = \frac{1}{5}$

$\mu_{\Phi}(\tau_3) = \frac{1}{\tau_0}$

Min. feasible with RH
Theorem (Liu, Layland, 1973): RM is optimal among all fixed-priority scheduling algorithms.

Def.: The response time $R_{i,j}$ of an instance $j$ of task $i$ is the time (measured from the arrival time) at which the instance is finished: $R_{i,j} = f_{i,j} - a_{i,j}$.

The critical instant of a task is the time at which the arrival of the task will produce the largest response time.
Response times and critical instants

- **Observation:**
  For RM, the critical instant $t$ of a task $\tau_i$ is given by the time when $\tau_{i,j}$ arrives together with all tasks $\tau_1, \ldots, \tau_{i-1}$ with higher priority.
Response times and critical instants

- For our “worst case task sets” we can assume that there are critical instants where an instance of a task arrives together with all higher priority tasks.
- A task set is schedulable, if the response time at these critical instants is not larger than the relative deadline.
Non-RM Schedule

Schedule feasible iff $C_1 + C_2 \leq T_1$
RM-Schedule

- Let $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$ be the number of periods of $\tau_1$ entirely contained in $T_2$.

- Case 1:
  
  - The computation time $C_1$ is short enough, so that all requests of $\tau_1$ within period of $\tau_2$ are completed before second request of $\tau_2$.
  - I.e. $C_1 \leq T_2 - FT_1$
  - Schedule feasible if $(F+1)C_1 + C_2 \leq T_2$
RM-Schedule

- Case 2:
  - The second request of $\tau_2$ arrives when $\tau_1$ is running.
  - i.e. $C_1 \geq T_2 - FT_1$

Schedule feasible if $FC_1 + C_2 \leq FT_1$
Proof of Liu/Layland