IN4343 Real-Time Systems

Aperiodic Task Scheduling
Recall

**Schedule**

Is a particular assignment of tasks to the processor that determines the task execution sequence:

Formally, given a task set \( \Gamma = \{\tau_1, ..., \tau_n\} \), a schedule is a function \( \sigma: \mathbb{R}^+ \rightarrow \mathbb{N} \) that associates an integer \( k \) to each interval of time \([t, t+1]\) with the following meaning:

\[
\begin{align*}
    k = 0 & \quad \text{in } [t, t+1] \text{ the processor is IDLE} \\
    k > 0 & \quad \text{in } [t, t+1] \text{ the processor executes } \tau_k
\end{align*}
\]
Definitions

• A schedule $\sigma$ is said to be feasible if all the tasks are able to complete within a set of constraints.

• A set of tasks $\Gamma$ is said to be schedulable if there exists a feasible schedule for it.
The general scheduling problem

Given a set $\Gamma$ of $n$ tasks, a set $P$ of $m$ processors, and a set $R$ of $r$ resources, find an assignment of $P$ and $R$ to $\Gamma$ which produces a feasible schedule.
Simplifying assumptions

- Single processor
- Homogeneous task sets
- Fully preemptive tasks
- Simultaneous activations
- No precedence constraints
- No resource constraints
Algorithm taxonomy

- Preemptive vs. Non Preemptive
- Static vs. dynamic
- On line vs. Off line
- Optimal vs. Heuristic
Optimality criteria

- **Feasibility**: Find a feasible schedule if there exists one
- Minimize the maximum lateness
- Minimize the number of deadline misses
- Assign a value to each task, then maximize the value of the feasible tasks
Graham’s Notation

\[ \alpha \mid \beta \mid \gamma \]

\begin{align*}
\alpha & \text{ denotes the number of processors} \\
\beta & \text{ denotes the constraints on tasks} \\
\gamma & \text{ denotes the optimality criterion}
\end{align*}

Examples:

1 \hspace{0.5cm} \text{preem.} \hspace{0.5cm} R_{\text{avg}} \hspace{3cm} \text{SJF - Uniprocessor algorithm for preemptive tasks that minimizes the average response time.}

1 \hspace{0.5cm} \text{sync} \hspace{0.5cm} L_{\text{max}} \hspace{3cm} \text{EDD - Uniprocessor algorithm for synchronous tasks that minimizes the maximum lateness.}

1 \hspace{0.5cm} \text{preem.} \hspace{0.5cm} L_{\text{max}} \hspace{3cm} \text{EDF - Uniprocessor algorithm for preemptive tasks that minimizes the maximum lateness.}
Classical scheduling policies

- First Come First Served
- Shortest Job First
- Priority Scheduling
- Round Robin

Not suited for real-time systems
First Come First Served

It assigns the CPU to tasks based on their arrival times.

- Non preemptive
- Dynamic
- On line
- Not optimal
First Come First Served

- Very unpredictable

response times strongly depend on task arrivals.

![Diagram showing task arrival times and response times with R_1 = 20, R_2 = 26, R_3 = 26, and another set of R_1 = 26, R_2 = 8, R_3 = 2]
Shortest Job First (SJF)

It selects the task with the shortest computation time.

- Non preemptive or preemptive
- Static \((C_i)\) is a constant parameter
- It can be used on line or off-line
- It minimizes the average response time
SJF Optimality

\[ f_{S'} + f_{L'} \leq f_{S} + f_{L} \]

\[ \bar{R}(\sigma') = \frac{1}{n} \sum_{i=1}^{n} (f'_{i} - r_{i}) \leq \frac{1}{n} \sum_{i=1}^{n} (f_{i} - r_{i}) = \bar{R}(\sigma) \]
SJF Optimality

\( \sigma \rightarrow \sigma' \rightarrow \sigma'' \rightarrow \ldots \rightarrow \sigma^* \)

\[ \bar{R}(\sigma) \geq \bar{R}(\sigma') \geq \bar{R}(\sigma'') \ldots \geq \bar{R}(\sigma^*) \]

\( \sigma^* = \sigma_{SJF} \)

\( \bar{R}(\sigma_{SJF}) \) is the minimum response time achievable by any algorithm
Is SJF suited for Real-Time?

- It is not optimal in the sense of feasibility.
Priority Scheduling

- Each task is assigned a priority: $p_i \in [0, 255]$
- The task with the highest priority is selected for execution.
- Tasks with the same priority are served FCFS.
  - Preemptive
  - Static or dynamic
  - On line
Priority Scheduling

- Problem: starvation
  low priority tasks may experience long delays due to the preemption of high priority tasks.

- A solution: aging
  priority increases with waiting time

**NOTE:**

\[
p_i \propto \frac{1}{C_i} \quad \Rightarrow \quad \text{SJF} \\
p_i \propto \frac{1}{r_i} \quad \Rightarrow \quad \text{FCFS}
\]
Round Robin

- The ready queue is served as FCFS, but ...
- Each task $\tau_i$ cannot execute more than $Q$ time units ($Q =$ time quantum).
- When $Q$ expires, $\tau_i$ is put back in the queue.
Round Robin

$n = \text{number of tasks in the system}$

\[ R_i \approx (nQ) \frac{C_i}{Q} = nC_i \]

**Time sharing**

Each task runs as if it was executing alone on a virtual processor $n$ times slower than the real one.
Round Robin

- if \( Q > \max(C_i) \) then \( RR \equiv FCFS \)

- if \( Q \approx \text{context switch time } (\delta) \) then

\[
R_i \approx n(Q + \delta) \frac{C_i}{Q} = nC_i\left(\frac{Q + \delta}{Q}\right)
\]
Real-Time Algorithms

Tasks can be scheduled by

- relative deadlines $D_i$ \hspace{1cm} (static)
- absolute deadlines $d_i$ \hspace{1cm} (dynamic)
Earliest Due Date

It selects the task with the earliest relative deadline [Jackson’ 55].

- All tasks arrive simultaneously
- Fixed priority \((D_i \text{ is known in advance})\)
- Preemption is not an issue
- It minimizes the maximum lateness \(L_{\text{max}}\)
Lateness

\[ L_i = f_i - d_i \]

\[ L_i > 0 \]

\[ L_i < 0 \]
Maximum Lateness

\[ L_{\text{max}} = \max_i (L_i) \]

if \((L_{\text{max}} < 0)\) then

no task misses its deadline
Exercise (3 min)

Check whether EDD produces a feasible schedule

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>$d_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>t2</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>t3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>t4</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the maximum lateness?
EDD Optimality

\[ L_{\text{max}} = L_a = f_a - d_a \]

\[ L'_a = f'_a - d_a < f_a - d_a \]

\[ L'_b = f'_b - d_b < f_a - d_a \]

\[ L'_{\text{max}} < L_{\text{max}} \]
EDD Optimality

$\sigma \rightarrow \sigma' \rightarrow \sigma'' \rightarrow \ldots \rightarrow \sigma^*$

$L_{\text{max}}(\sigma) \geq L_{\text{max}}(\sigma') \geq L_{\text{max}}(\sigma'') \ldots \geq L_{\text{max}}(\sigma^*)$

$\sigma^* = \sigma_{\text{EDD}}$

$L_{\text{max}}(\sigma_{\text{EDD}})$ is the minimum value achievable by any algorithm.
EDD - Guarantee test (off line)

A task set $\Gamma$ is feasible iff $\forall i \ f_i \leq d_i$

$$f_i = \sum_{k=1}^{i} C_k$$

$$\forall i \ \sum_{k=1}^{i} C_k \leq D_i$$
Earliest Deadline First

It selects the task with the earliest absolute deadline [Horn 74].

- Tasks may arrive at any time
- Dynamic priority \( (d_i \) depends on arrival)\)
- Full preemptive tasks
- It minimizes the maximum lateness \( (L_{\text{max}}) \)
EDF Example
EDF Guarantee test (on line)

∀i \sum_{k=1}^{i} c_k(t) \leq d_i - t
Check whether EDF produces a feasible schedule

<table>
<thead>
<tr>
<th></th>
<th>( C_i )</th>
<th>( D_i )</th>
<th>( r_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>4</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>t2</td>
<td>5</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>t3</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>t4</td>
<td>3</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

What is the maximum lateness?
Complexity Issues

**EDD**
- $O(n \log n)$ to order the task set
- $O(n)$ to guarantee the whole task set

**EDF**
- $O(n)$ to insert a new task in the queue
- $O(n)$ to guarantee a new task
EDF optimality
(in the sense of feasibility)

[Dertouzos 1974]

An algorithm A is **optimal** in the sense of feasibility if it generates a feasible schedule, if there exists one.

**Demonstration method**

It is sufficient to prove that, given an arbitrary feasible schedule, the schedule generated by EDF is also feasible.
Dertouzos Transformation

\[
\begin{align*}
\sigma(t) &= \text{task executing at time } t \\
E(t) &= \text{task with min } d \text{ at time } t \\
t_E &= \text{time at which } E \text{ is executed}
\end{align*}
\]

\[
\sigma \neq \sigma_{\text{EDF}}
\]

\[
\text{for } (t = 0 \text{ to } D_{\text{max}} - 1) \quad \text{if } (\sigma(t) \neq E(t)) \quad \{
\begin{align*}
\sigma(t_E) &= \sigma(t); \\
\sigma(t) &= E(t);
\end{align*}
\}
\]

\[
\begin{align*}
\sigma(t) &= 4 \\
E(t) &= 2 \\
t_E &= 6
\end{align*}
\]
Dertouzos Transformation

Dertouzos transformation algorithm preserves schedulability, in fact:

- this is obvious for the advanced slice!
- for the postponed slice, the slack cannot decrease:

\[ t = 5 \]
A property of optimal algorithms

If an optimal algorithm (in the sense of feasibility) produces an unfeasible schedule, then no algorithm can do that.

If an algorithm $A$ minimizes $L_{\text{max}}$ then $A$ is also optimal in the sense of feasibility. The opposite is not true.
Non Preemptive Scheduling

Under non preemptive execution, EDF is not optimal:

Feasible schedule

EDF
Non Preemptive Scheduling

To achieve optimality, an algorithm should be clairvoyant, and decide to leave the CPU idle in the presence of ready tasks:

If we forbid to leave the CPU idle in the presence of ready tasks, then EDF is optimal.

NP-EDF is optimal among non-idle scheduling algorithms
Non preemptive scheduling algorithms

The problem of finding a feasible schedule is **NP hard** and is treated **off line** with tree search algorithms:

- depth = $n$
- \# leaves = $n!$
- complexity: $O(n \cdot n!)$
Bratley’s Algorithm

\[(1 \mid \text{no-preem} \mid L_{\text{max}})\]

Reduces the average complexity by pruning techniques:

Do not expand unless the partial schedule is found to be **strongly feasible**.

A partial schedule is said to be **strongly feasible** if adding any of the remaining nodes it remains feasible.
Bratley’s Algorithm \(( 1 \mid \text{no-preem} \mid L_{\text{max}})\)

Reduces the average complexity by pruning techniques:

**Example**

<table>
<thead>
<tr>
<th>(a_i)</th>
<th>(C_i)</th>
<th>(d_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(\tau_4)</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

```
\sigma_1 = \{4, 2, 3, 1\} \\
\sigma_2 = \{4, 3, 2, 1\}
```
Exercise (7 min)

Book: 3.3

Find feasible schedules using Bratley’s algorithm

<table>
<thead>
<tr>
<th>t</th>
<th>a_i</th>
<th>C_i</th>
<th>D_i</th>
<th>d_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>0</td>
<td>6</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>t2</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>t3</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>t4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>16</td>
</tr>
</tbody>
</table>
Example of heuristic algorithm

Spring algorithm [Stankovic & Ramamritham 87]

1. The schedule for a set of N tasks is constructed in N steps
2. The search is driven by a heuristic function H
3. At each step the algorithm selects the task that minimizes the heuristic function

\[
\text{Backtracking is possible}
\]
Example of heuristic algorithm

Spring algorithm [Stankovic & Ramamritham 87]

Example of heuristic functions:

\[ H = r_i \quad \Rightarrow \quad \text{FCFS} \]
\[ H = C_i \quad \Rightarrow \quad \text{SJF} \]
\[ H = D_i \quad \Rightarrow \quad \text{DM} \]
\[ H = d_i \quad \Rightarrow \quad \text{EDF} \]

Composit heuristic functions:

\[ H = w_1 r_i + w_2 D_i \]
\[ H = w_1 C_i + w_2 d_i \]
Exercise (5 min)

Book: 3.4

Find feasible schedules using Spring algorithm with

\[ H = a + C + D \]

<table>
<thead>
<tr>
<th></th>
<th>( a_i )</th>
<th>( C_i )</th>
<th>( D_i )</th>
<th>( d_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>0</td>
<td>6</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>t2</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>t3</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>t4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>16</td>
</tr>
</tbody>
</table>

Can you find a better heuristic function?
Example of heuristic algorithm

Spring algorithm [Stankovic & Ramamritham 87]

Possibility to handle precedence constraints:

Eligibility

\[ \tau_i \]

\[ E_i = \infty \quad E_i = 1 \]

Heuristic functions:

\[
H = E_i \left( w_1 r_i + w_2 D_i \right)
\]

\[
H = E_i \left( w_1 C_i + w_2 d_i \right)
\]
Example of heuristic algorithm

Spring algorithm [Stankovic & Ramamritham 87]

Complexity:
\[
\begin{align*}
\text{Exhaustive search:} & \quad O(N \cdot N!) \\
\text{Heuristic search:} & \quad O(N^2) \\
\text{Heuristic w. k btracks:} & \quad O(kN^2)
\end{align*}
\]
Example of heuristic algorithm

Spring algorithm [Stankovic & Ramamritham 87]

If a feasible schedule is not found, does not mean that there does not exist one.

If a feasible solution is found, the schedule is stored into a dispatch list:

- next
- Task ID
- start time
- length
- Ø
Scheduling with precedence constraints

1 | prec, sync | $L_{\text{max}}$

**Latest Deadline First (LDF)** \cite{Lawler73}

Given a precedence graph, it constructs the schedule from the tail: among the nodes with no successors, LDF selects the task with the latest deadline:

![Precedence Graph and Schedules]

- **LDF** schedule:
  - Task A is scheduled first, followed by B, D, C, E, and F.
  - D misses its deadline.

- **EDF** schedule:
  - Tasks A, C, B, D, E, and F are scheduled without deadline violations.
Latest Deadline First

Proof of optimality

- Start from optimal non-LDF schedule $\sigma$
  - $\Gamma$: set of all tasks without successors
  - $J_l$: job $l \in \Gamma$ with latest deadline $d_l$
  - $J_k$: last job in $\sigma$ with deadline $d_k$ ($\leq d_l$)

• Create new schedule $\sigma^*$ by moving $J_l$ to the back
  - $L_{\text{max}}(\sigma^*) \leq L_{\text{max}}(\sigma)$
  - remove $J_l$ from $\sigma^*$ and repeat
Scheduling with precedence constraints

1 | prec, preem | $L_{\text{max}}$

EDF* [Chetto & Chetto 89]

- Assumes that arrival times are known a priori;
- Transforms precedence constraints into timing constraints by modifying arrival times and deadlines based on the precedence graph;
- Applies EDF to the modified task set.
Scheduling with precedence constraints

EDF* [Chetto & Chetto 89] 1 | prec, preem | $L_{\text{max}}$

The idea is to:

⇒ postpone the arrival time of a successor: $r^*_B = r_A + C_A$

⇒ advance the deadline of a predecessor: $d^*_A = d_B - C_B$
Scheduling with precedence constraints

EDF* [Chetto & Chetto 89] 1 | prec, preem | L_max

Arrival time modification

1. For all root nodes, set \( r^*_i = r_i \).

2. Select a task \( \tau_i \) such that all its immediate predecessors have been modified, else exit.

3. Set \( r^*_i = \max \{ r_i, \max_{\tau_k \rightarrow \tau_i} (r^*_k + C_k) \} \).

4. Repeat from line 2.
Scheduling with precedence constraints

EDF* [Chetto & Chetto 89]  1 | prec, preem | $L_{\text{max}}$

Deadline modification

1. For all leaves, set $d^*_i = d_i$.

2. Select a task $\tau_i$ such that all its immediate successors have been modified, else exit.

3. Set $d^*_i = \min \{ d_i, \min_{\tau_i \rightarrow \tau_k} (d^*_k - C_k) \}$.

4. Repeat from line 2.
Exercise (5 min)

Book: 3.5

Modify release times and deadlines to schedule by EDF taking precedence relations into account

<table>
<thead>
<tr>
<th></th>
<th>( r_i )</th>
<th>( C_i )</th>
<th>( d_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>5</td>
<td>25</td>
</tr>
</tbody>
</table>

A \( \rightarrow \) C
B \( \rightarrow \) C
B \( \rightarrow \) D
C \( \rightarrow \) E
C \( \rightarrow \) F
D \( \rightarrow \) F
D \( \rightarrow \) G
EDF with precedence constraints

**Proof of optimality**

Transformation ensures that
- \( r^*_i \geq r_i \)
- \( d^*_i \leq d_i \)

Therefore if modified task set is schedulable with EDF, then so was the original.

Check that precedencies are respected
- \( \tau_u \rightarrow \tau_v \)
- \( d^*_u < d^*_v \) so \( \tau_v \) has lower priority, hence cannot preempt \( \tau_u \), and cannot start earlier as \( r^*_u < r^*_v \)
## Summary

<table>
<thead>
<tr>
<th>activ</th>
<th>prec</th>
<th>preem</th>
<th>algorithm</th>
<th>authors</th>
<th>compl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>N</td>
<td>*</td>
<td>EDD</td>
<td>Jackson '55</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>Asyn</td>
<td>N</td>
<td>Y</td>
<td>EDF</td>
<td>Horn '74</td>
<td>$n \forall \text{task}$</td>
</tr>
<tr>
<td>Asyn</td>
<td>N</td>
<td>N</td>
<td>NP-EDF</td>
<td>Jeffay '91</td>
<td>$n \forall \text{task}$</td>
</tr>
<tr>
<td>Asyn</td>
<td>Y</td>
<td>N</td>
<td>Tree search</td>
<td>Bratley '71</td>
<td>$n \cdot n!$</td>
</tr>
<tr>
<td>Asyn</td>
<td>Y</td>
<td>N</td>
<td>Spring</td>
<td>Stankovic '87</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Sync</td>
<td>Y</td>
<td>*</td>
<td>LDF</td>
<td>Lawler '73</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>Asyn</td>
<td>Y</td>
<td>Y</td>
<td>EDF*</td>
<td>Chetto '89</td>
<td>$n \log n$</td>
</tr>
</tbody>
</table>