Aperiodic Task Scheduling

Radek Pelánek
Preemptive Scheduling: The Problem

- 1 processor
- arbitrary arrival times of tasks
- preemption
- performance measure: maximum lateness
- no resources, no precedence constraints
Example

<table>
<thead>
<tr>
<th></th>
<th>J₁</th>
<th>J₂</th>
<th>J₃</th>
<th>J₄</th>
<th>J₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>aᵢ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Cᵢ</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>dᵢ</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>

1. Solve the example (manually).
2. Try to find out a scheduling algorithm.
Earliest Deadline First Algorithm

EDF

At any instant execute the task with the earliest absolute deadline among all the ready tasks.
Example – EDF Schedule

<table>
<thead>
<tr>
<th></th>
<th>a_i</th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_i</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>d_i</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Earliest Deadline First
Theorem (Horn)

Given a set of $n$ independent tasks with arbitrary arrival times, the EDF algorithm is optimal with respect to minimizing the maximum lateness.
Proof of Horn’s Theorem

Basic idea of the proof:

1. Let $\sigma$ be an optimal schedule; we transform it into an EDF schedule $\sigma_{EDF}$ without increasing maximum lateness.
2. The schedule is divided into time slices of 1 unit.
3. Transformation: interchange 1 appropriate time slice.
Preemptive Scheduling

Non-preemptive Scheduling

Precedence Constraints

Summary

Earliest Deadline First

Proof of Horn's Theorem (cont.)

Figure 3.4  Proof of the optimality of the EDF algorithm. a. schedule $\sigma$ at time $t = 4$. b. new schedule obtained after a transposition.
Guarantee Test

- is the set of tasks schedulable?
- sort tasks by increasing deadlines
- synchronous activation \((a_i = 0)\) – schedulability guaranteed by conditions:

\[
\forall i : \sum_{k=1}^{i} C_k \leq d_i
\]

- asynchronous activation: “dynamical version” of the previous test
EDF is Fine ...

EDF:
- is optimal
- works on-line
- easy to implement
- simple guarantee test
... but Not a Silver Bullet

- EDF is not optimal with more than 1 processor
- Try to find a specific set of tasks:
  - the set is schedulable on 2 processor
  - EDF schedule misses some deadline
EDF with More Processors

2 Processors

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$d_i$</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Schedulable, but EDF schedule misses deadline for $J_3$. 
Least Slack Time Algorithm

LST

At any instant execute the task with the least slack time (that is $d_i - C_i$) among all the ready tasks.

LST is also optimal.
Example

Find a set of task such that EDF and LST produce different schedules.
Non-preemptive Scheduling: The Problem

- 1 processor
- arbitrary arrival times of tasks
- preemption not allowed
- performance measure: maximum lateness
- no resources, no precedence constraints
Non-optimality of EDF

EDF is not optimal for non-preemptive scheduling.

Find a set of task such that EDF does not produce optimal schedule.
Non-optimality of EDF

<table>
<thead>
<tr>
<th>J_1</th>
<th>J_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_i</td>
<td>0</td>
</tr>
<tr>
<td>C_i</td>
<td>4</td>
</tr>
<tr>
<td>d_i</td>
<td>7</td>
</tr>
</tbody>
</table>

Optimal schedule vs EDF schedule.
EDF and Non-idle Schedules

- **non-idle** algorithm - does not permit the processor to be idle when there are active jobs
- restriction to non-idle algorithms $\Rightarrow$ EDF is optimal
Optimal Scheduling

- no on-line algorithm can generate optimal schedule (example above: time 0, stay idle or start $J_1$?)
- off-line: non-preemptive scheduling is NP-complete
- branch-and-bound (backtracking) algorithms, worst case complexity $O(n \cdot n!)$
Search Tree

- Empty schedule
- Partial schedule
- Feasible schedule
- Complete schedule
Reminder: Backtracking, $n$ Queen Problem
Pruning

- branch is abandoned when:
  - the addition of any node to the current path causes a missed deadline
  - a feasible schedule is found at the current path
- size of the search tree is exponential in the worst case
- significant pruning in the average case
Pruning: Example

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$C_i$</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$d_i$</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

Number in the node = scheduled task
Number outside the node = finishing time

$J_i^+$ = task that misses its deadline

○ = feasible schedule
Example

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>$C_i$</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$d_i$</td>
<td>18</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Draw the search tree (with pruning).
Scheduling with Precedence Constraints

- generally NP-complete
- we consider two special cases, polynomial algorithms
- remark on heuristical approach
Precedence Constraints: Problem 1

- 1 processor
- precedence constraints
- synchronous activation \((\forall i : a_i = 0)\)
- (preemption does not matter)
- performance measure: maximum lateness
Example

1. solve the example (manually)
2. construct the EDF schedule
3. try to find out an optimal scheduling algorithm
one possible solution: *latest deadline first* (LDF)

- note: different interpretations
  - EDF algorithm = *earliest deadline* job is scheduled to run *first*
  - LDF algorithm = *latest deadline* job is put into schedule *first*
Latest Deadline First Algorithm

LDF

- among tasks without successors select the task with the latest deadline
- remove this task from the precedence graph and put it into a stack
- repeat until all tasks are in the stack
- the stack represents the order in which tasks should be scheduled

LDF is optimal.
**LDF: example**

**Table:**

<table>
<thead>
<tr>
<th></th>
<th>J₁</th>
<th>J₂</th>
<th>J₃</th>
<th>J₄</th>
<th>J₅</th>
<th>J₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cᵢ</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>dᵢ</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

**DAG:**

- J₁ → J₂ → J₃ → J₄ → J₅ → J₆
- J₁ → J₄
- J₁ → J₅
- J₁ → J₆

**LDF Diagram:**

Lₘₐₓ = 0

**EDF Diagram:**

Lₘₐₓ = L₄ = 1
Precedence Constraints: Problem 2

- 1 processor
- precedence constraints
- arbitrary arrival times of tasks
- preemption
- performance measure: maximum lateness
Example

Precedence constraints:

\[ A \rightarrow C; A \rightarrow D; B \rightarrow D, B \rightarrow E; E \rightarrow D; C \rightarrow F; D \rightarrow F \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a</strong></td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td><strong>d</strong></td>
<td>8</td>
<td>8</td>
<td>13</td>
<td>10</td>
<td>5</td>
<td>14</td>
</tr>
</tbody>
</table>
Basic Idea

1. Transform set $J$ of dependent tasks into set $J^*$ of independent tasks.
2. Apply EDF to set $J^*$.

Transformation done by modification of arrival times and deadline times.
Modification of Arrival Times

If $J_Y \rightarrow J_X$ then:

- $s_X \geq a_X$
  $J_X$ cannot start earlier than its activation time

- $s_X \geq a_Y + C_Y$
  $J_X$ cannot start earlier than the minimum finishing time of $J_Y$
Modification of Arrival Times (cont.)

new arrival time:

\[ a_X^* = \max(a_X, \max(a_Y + C_Y, J_Y \rightarrow J_X)) \]

modified arrival time must be computed in the correct order
(given by precedence constraints)
If $J_X \rightarrow J_Y$ then:

- $f_X \leq d_X$
  - $J_X$ must finish within its deadline
- $f_X \leq d_Y - C_Y$
  - $J_X$ must finish not later than the maximum starting time of $J_Y$
Modification of Deadlines (cont.)

new deadline:

\[ d_X^* = \min(d_X, \min(d_Y - C_Y, J_X \rightarrow J_Y)) \]

modified deadline times must be computed in the correct order (given by precedence constraints)
Theorem

There exists feasible schedule for the modified task set $J^*$ under EDF if and only if the original task set is schedulable.
Example

Precedence constraints:
A → C; A → D; B → D, B → E; E → D; C → F; D → F

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>8</td>
<td>8</td>
<td>13</td>
<td>10</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>a*</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>d*</td>
<td>7</td>
<td>4</td>
<td>11</td>
<td>10</td>
<td>5</td>
<td>14</td>
</tr>
</tbody>
</table>
Example

- precedence constraints: $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow E$, $D \rightarrow F$, $B \rightarrow D$, $C \rightarrow F$, $D \rightarrow G$
- all tasks: $a_i = 0$, $D_i = 25$, computation times: 2, 3, 3, 5, 1, 2, 5
- compute modified arrival times and deadlines
- compute schedule according to EDF
Heuristical search

- more complicated problems (e.g., non-preemptive, precedence constraints) NP-complete
- brute-force search (with pruning)
- heuristical search
  - only some schedules are considered
  - no guarantee of optimality
- note: general computer science approach
Heuristical search

- constructs partial schedules (like brute-force search)
- considers only one (or few) tasks for extension of a schedule
- selection based on heuristic function $H$
Heuristic function

\[ H(i) = a_i \quad \text{first come first serve} \]
\[ H(i) = C_i \quad \text{shortest job first} \]
\[ H(i) = d_i \quad \text{earliest deadline first} \]

- more complicated parameters (taking into account precedence relations, resources, ...)
- weighted combinations of different parameters
### Overview of Problems and Algorithms

<table>
<thead>
<tr>
<th></th>
<th>sync. act.</th>
<th>preemptive act.</th>
<th>non-preemptive act.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>independent</strong></td>
<td>EDF, $O(n \log n)$</td>
<td>EDF, LST, $O(n^2)$</td>
<td>tree search, $O(n \cdot n!)$</td>
</tr>
<tr>
<td><strong>precedence</strong></td>
<td>LDF, $O(n^2)$</td>
<td>modified EDF, $O(n^2)$</td>
<td>heuristic search</td>
</tr>
</tbody>
</table>
Summary

- aperiodic task scheduling (arbitrary arrival times)
- 1 processor, no resources
- precedence constraints: yes/no
- preemption: yes/no
- performance measure: maximum lateness
- algorithms: earliest deadline first (EDF), least slack time (LST), branch and bound, latest deadline first (LDF), transformations