Classification - Examples

- $1|r_j|C_{max}$
  
  - given: $n$ jobs with processing times $p_1, \ldots, p_n$ and release dates $r_1, \ldots, r_n$
  
  - jobs have to be scheduled without preemption on one machine taking into account the earliest starting times of the jobs, such that the makespan is minimized
  
  - $n = 4$, $p = (2, 4, 2, 3)$, $r = (5, 4, 0, 3)$
Classification - Examples

- $1|r_j|C_{max}$
  - given: $n$ jobs with processing times $p_1, \ldots, p_n$ and release dates $r_1, \ldots, r_n$
  - jobs have to be scheduled without preemption on one machine taking into account the earliest starting times of the jobs, such that the makespan is minimized
  - $n = 4$, $p = (2, 4, 2, 3)$, $r = (5, 4, 0, 3)$

Feasible Schedule with $C_{max} = 12$ (schedule is optimal)
Classification - Examples

- \( F^2 || \sum w_j T_j \)
  
  - given \( n \) jobs with weights \( w_1, \ldots, w_n \) and due dates \( d_1, \ldots, d_n \)
  
  - operations \((i, j)\) with processing times \( p_{ij}, i = 1, 2; j = 1, \ldots, n \)
  
  - jobs have to be scheduled on two machines such that operation \((2, j)\) is scheduled on machine 2 and does not start before operation \((1, j)\), which is scheduled on machine 1, is finished and the total weighted tardiness is minimized

- \( n = 3, \; p = \begin{pmatrix} 2 & 1 & 3 \\ 3 & 4 & 1 \end{pmatrix}, \; w = (3, 1, 5), \; d = (6, 8, 4) \)
Classification - Examples

- $F^2|| \sum w_j T_j$
  - given $n$ jobs with weights $w_1, \ldots, w_n$ and due dates $d_1, \ldots, d_n$
  - operations $(i, j)$ with processing times $p_{ij}$, $i = 1, 2; j = 1, \ldots, n$
  - jobs have to be scheduled on two machines such that operation $(2, j)$ is scheduled on machine 2 and does not start before operation $(1, j)$, which is scheduled on machine 1, is finished and the total weighted tardiness is minimized

- $n = 3$, $p = \begin{pmatrix} 2 & 1 & 3 \\ 3 & 4 & 1 \end{pmatrix}$, $w = (3, 1, 5)$, $d = (6, 8, 4)$

\[
\sum w_j T_j = 3(8 - 6) + (12 - 9) + 5(4 - 4) = 9
\]
Classes of Schedules

- Nondelay Schedules:
  A feasible schedule is called a nondelay schedule if no machine is kept idle while a job/an operation is waiting for processing

Example: \( P3|prec|C_{max} \)

\[ n = 6 \]
\[ p = (1, 1, 2, 2, 3, 3) \]
Classes of Schedules

- Nondelay Schedules:
  A feasible schedule is called a nondelay schedule if no machine is kept idle while a job/an operation is waiting for processing.

Example: $P3|\text{prec}|C_{max}$

$n = 6$

$p = (1, 1, 2, 2, 3, 3)$

Best nondelay:

<table>
<thead>
<tr>
<th></th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Optimal:

<table>
<thead>
<tr>
<th></th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Classes of Schedules

Remark: restricted to non preemptive schedules

- Active Schedules:
  A feasible schedule is called active if it is not possible to construct another schedule by changing the order of processing on the machines and having at least one job/operation finishing earlier and no job/operation finishing later.

- Semi-Active Schedules:
  A feasible schedule is called semi-active if no job/operation can be finishing earlier without changing the order of processing on any one of the machines.
Classes of Schedules

Examples of (semi)-active schedules:

Prec: $1 \rightarrow 2; \ 2 \rightarrow 3$

not semi-active

<table>
<thead>
<tr>
<th>M1</th>
<th>1</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Classes of Schedules

Examples of (semi)-active schedules:

Prec: 1 → 2; 2 → 3

not semi-active

\[
\begin{array}{c|c|c}
M1 & 1 & 3 & 5 \\
M2 & 2 & 4 & \\
\end{array}
\]

semi-active

\[
\begin{array}{c|c|c}
M1 & 1 & 3 & 5 \\
M2 & 2 & 4 & \\
\end{array}
\]
Classes of Schedules

Examples of (semi)-active schedules:

Prec: $1 \rightarrow 2; \ 2 \rightarrow 3$

not semi-active

<table>
<thead>
<tr>
<th>M1</th>
<th>1</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

semi-active

<table>
<thead>
<tr>
<th>M1</th>
<th>1</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

active

<table>
<thead>
<tr>
<th>M1</th>
<th>1</th>
<th>5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Classes of Schedules

Properties:

• every nonpreemptive nondelay schedule is active
• every active schedule is semiactive
• if the objective criterion is regular, the set of active schedules contains an optimal schedule (regular = non decreasing with respect to the completion times)

Summary:
Research topics for Scheduling

- determine boarder line between polynomially solvable and NP-hard models
- for polynomially solvable models
  - find the most efficient solution method (low complexity)
- for NP-hard models
  - develop enumerative methods (DP, branch and bound, branch and cut, ...)
  - develop heuristic approached (priority based, local search, ...)
  - consider approximation methods (with quality guarantee)
Intermezzo: Complexity Theory

- mathematical framework to study the difficulty of algorithmic problems

Notations/Definitions

- problem: generic description of a problem (e.g. 1|| $\sum C_j$)
- instance of a problem: given set of numerical data (e.g. $n, p_1, \ldots, p_n$)
- size of an instance $I$: length of the string necessary to specify the data (Notation: $|I|$)
  - binary encoding: $|I| = n + \log(p_1) + \ldots + \log(p_n)$
  - unary encoding: $|I| = n + p_1 + \ldots + p_n$
Intermezzo: Complexity Theory

Notations/Definitions

- efficiency of an algorithm: upper bound on number of steps depending on the size of the instance (worst case consideration)

- big O-notation: for an \( O(f(n)) \) algorithm a constant \( c > 0 \) and an integer \( n_0 \) exist, such that for an instance \( I \) with size \( n = |I| \) and \( n \geq n_0 \) the number of steps is bounded by \( cf(n) \)

  Example: \( 7n^3 + 230n + 10 \log(n) \) is \( O(n^3) \)

- (pseud)polynomial algorithm: \( O(p(|I|)) \) algorithm, where \( p \) is a polynomial and \( I \) is coded binary (unary)

  Example: an \( O(n \log(\sum p_j)) \) algorithm is a polynomial algorithm and an \( O(n \sum p_j) \) algorithm is a pseudopolynomial algorithm
Intermezzo: Complexity Theory

Classes \( \mathcal{P} \) and \( \mathcal{NP} \)

- A problem is (pseudo)polynomial solvable if a (pseudo)polynomial algorithm exists which solves the problem.
- Class \( \mathcal{P} \): contains all decision problems which are polynomial solvable.
- Class \( \mathcal{NP} \): contains all decision problems for which - given an ’yes’ instance - the correct answer, given a proper clue, can be verified by a polynomial algorithm.

Remark: each optimization problem has a corresponding decision problem by introducing a threshold for the objective value (does a schedule exist with objective smaller \( k \)?)
Intermezzo: Complexity Theory

Polynomial reduction

- a decision problem $P$ polynomially reduces to a problem $Q$, if a polynomial function $g$ exists that transforms instances of $P$ to instances of $Q$ such that $I$ is a 'yes' instance of $P$ if and only if $g(I)$ is a 'yes' instance of $Q$

Notation: $P \preceq Q$

NP-complete

- a decision problem $P \in \mathcal{NP}$ is called NP-complete if all problems from the class $\mathcal{NP}$ polynomially reduce to $P$
- an optimization problem is called NP-hard if the corresponding decision problem is NP-complete
Intermezzo: Complexity Theory

Examples of NP-complete problems:

- **SATISFIABILITY**: decision problem in Boolean logic, Cook in 1967 showed that all problems from $\mathcal{NP}$ polynomially reduce to it

- **PARTITION**:
  - given $n$ positive integers $s_1, \ldots, s_n$ and $b = 1/2 \sum_{j=1}^{n} s_j$
  - does there exist a subset $J \subset I = \{1, \ldots, n\}$ such that

$$\sum_{j \in J} s_j = b = \sum_{j \in I \setminus J} s_j$$
Intermezzo: Complexity Theory

Examples of NP-complete problems (cont.):

- **3-PARTITION**:  
  - given $3n$ positive integers $s_1, \ldots, s_{3n}$ and $b$ with $b/4 < s_j < b/2$, $j = 1, \ldots, 3n$ and $b = 1/n \sum_{j=1}^{3n} s_j$  
  - do there exist disjoint subsets $J_i \subset I = \{1, \ldots, 3n\}$ such that  
    $$\sum_{j \in J_i} s_j = b; \quad i = 1, \ldots, n$$
Intermezzo: Complexity Theory

Proofing NP-completeness

If an NP-complete problem $P$ can be polynomially reduced to a problem $Q \in \mathcal{NP}$, than this proves that $Q$ is NP-complete (transitivity of polynomial reductions)

Example: $\text{PARTITION} \propto P_2||C_{max}$
Proof: on the board

Famous open problem: Is $\mathcal{P} = \mathcal{NP}$?

- solving one NP-complete problem polynomially, would imply $\mathcal{P} = \mathcal{NP}$
Single machine models

Observation:

- for non-preemptive problems and regular objectives, a sequence in which the jobs are processed is sufficient to describe a solution

Dispatching (priority) rules

- static rules - not time dependent
  e.g. shortest processing time first, earliest due date first

- dynamic rules - time dependent
  e.g. minimum slack first (slack = \( d_j - p_j - t \); \( t \) current time)

- for some problems dispatching rules lead to optimal solutions
Single machine models: $1\| \sum w_j C_j$

Given:

- $n$ jobs with processing times $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$

Consider case: $w_1 = \ldots = w_n (= 1)$:
Single machine models: $1|| \sum w_j C_j$

Given:

- $n$ jobs with processing times $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$

Consider special case: $w_1 = \ldots = w_n (= 1)$:

- SPT-rule: shortest processing time first
- **Theorem**: SPT is optimal for $1|| \sum C_j$
  Proof: by an exchange argument (on board)
- Complexity: $O(n \log(n))$
Single machine models: $1|| \sum w_j C_j$

General case

- **WSPT-rule:** weighted shortest processing time first, i.e. sort jobs by increasing $p_j/w_j$-values
- **Theorem:** WSPT is optimal for $1|| \sum w_j C_j$
  
  Proof: by an exchange argument (exercise)
- **Complexity:** $O(n \log(n))$

Further results:

- $1|tree| \sum w_j C_j$ can be solved by in polynomial time ($O(n \log(n))$)
  (see [Brucker 2004])
- $1|prec| \sum C_j$ is NP-hard in the strong sense
  (see [Brucker 2004])
Single machine models: $1|prec|f_{max}$

Given:

- $n$ jobs with processing times $p_1, \ldots, p_n$
- regular functions $f_1, \ldots, f_n$
- objective criterion $f_{max} = \max\{f_1(C_1), \ldots, f_n(C_n)\}$

Observation:

- completion time of last job $= \sum p_j$
Single machine models: $1|prec|f_{max}$

Given:

- $n$ jobs with processing times $p_1, \ldots, p_n$
- regular functions $f_1, \ldots, f_n$
- objective criterion $f_{max} = \max\{f_1(C_1), \ldots, f_n(C_n)\}$

Observation:

- completion time of last job $= \sum p_j$

Method

- plan backwards from $\sum p_j$ to 0
- from all available jobs (jobs from which all successors have already been scheduled), schedule the job which is 'cheapest' on that position
**Single machine models:** $1|\text{prec}|f_{\text{max}}$

- $S$ set of already scheduled jobs (initial: $S = \emptyset$)
- $J$ set of all jobs, which successors have been scheduled (initial: all jobs without successors)
- $t$ time where next job will be completed (initial: $t = \sum p_j$)

**Algorithm 1|\text{prec}|f_{\text{max}}$$ \text{(Lawler’s Algorithm)}$

**REPEAT**

select $j \in J$ such that $h_j(t) = \min_{k \in J} f_k(t)$;

schedule $j$ such that it completes at $t$;

add $j$ to $S$, delete $j$ from $J$ and update $J$;

$t := t - p_j$;

**UNTIL** $J = \emptyset$. 

**Single machine models:** 1|\textit{prec}|f_{max}

- **Theorem:** Algorithm 1|\textit{prec}|f_{max} is optimal for 1|\textit{prec}|f_{max}
  Proof: on the board

- **Complexity:** \(O(n^2)\)