Guest Lecture: Introduction to Scheduling

Thomas Lambert

April 23, 2018
A Small Example

- 2 computers and 6 programs to execute.
- I know the duration of each program, respectively 5s, 2s, 2s, 8s, 5s and 6s.
- I want to finish the complete execution as fast as possible (only the last computer to end is important).

```
r1
  
  r2
```
A Small Example

- 2 computers and 6 programs to execute.
- I know the duration of each program, respectively 5s, 2s, 2s, 8s, 5s and 6s.
- I want to finish the complete execution as fast as possible (only the last computer to end is important).

Keeping the order of the input: 16s
A Small Example

- 2 computers and 6 programs to execute.
- I know the duration of each program, respectively 5s, 2s, 2s, 8s, 5s and 6s.
- I want to finish the complete execution as fast as possible (only the last computer to end is important).

![Diagram showing task scheduling example]

Longest First: 15s, better!
A Small Example

- 2 computers and 6 programs to execute.
- I know the duration of each program, respectively 5s, 2s, 2s, 8s, 5s and 6s.
- I want to finish the complete execution as fast as possible (only the last computer to end is important).

The two longest ones on the same machine: 14s, best!
Scheduling is the distribution of activities (or jobs) on a limited amount of resources under some objectives.

Activities: programs to run, lectures, vehicles for public transportation, requests, steps of a construction project...
Resources: computers, workers, roads, rooms...
Objectives: Minimize completion time, memory usage, energy consumption, lateness, idle times...

In scheduling we study such problems and try to find algorithmic solutions.

In this lecture: presentation of some classic scheduling problems and some methodology approaches.
Sources

- Lecture greatly inspired by the ones of Loris Marchal (ENS de Lyon) and Peter Brucker (University of Osnabrueck).
- Many books to go further!

![Introduction to Scheduling](image1)

![Complex Scheduling](image2)
General Model

- General description of a scheduling problem:
  - Inputs: \( n \) jobs (set \( J \)), \( m \) machines (resources, set \( M \)) and a processing time \( p_i \) for every job \( i \).
  - Output: A schedule \( \sigma: J \rightarrow M \times \mathbb{R}^+ \).
    - \( \sigma(i) = (j, t) \): job \( i \) schedule on \( j \) and starting at time \( t \).
    - No overlapping jobs on a machine! If \( i \neq i' \), \( \sigma(i) = (j, t) \) and \( \sigma(i') = (j, t') \) then:
      \[
      t + p_i \leq t' \quad \text{or} \quad t' + p_{i'} \leq t.
      \]
- We are looking for algorithmic solutions.
- The more general is a problem, the harder it is to solve \( \rightarrow \) specific algorithms for specific variants.
A large part of scheduling problems can be described using a three fields classification.

Introduced by Graham et al. (1970).

$\alpha|\beta|\gamma$ with:
- $\alpha$ describing the resources environment.
- $\beta$ describing the jobs characteristics.
- $\gamma$ describing the objective nature.
1: Only one machine.

\( P: m \) parallel machines.
- Duration of a job \( i \): \( p_i \) for every machine \( j \).

\( Q: m \) related machines.
- Duration of a job \( i \): \( p_i/s_j \) on machine \( j \), \( s_j \) speed of the machine.

\( R: m \) unrelated machines
- Duration of a job \( i \): \( p_{i,j} \) on machine \( j \).

Multiple Stages (shop problems): each job is split into operations that have to be scheduled on particular machines.
Job Characteristics

- $r_i$: each job has a release date $r_i$ and can’t be scheduled before it ($\sigma(i) = (j, t)$ with $r_i \leq t$).
- $d_i$: each job has a due date $d_i$ and have to be processed before it ($\sigma(i) = (j, t)$ with $t + p_i \leq d_i$).
- **pmtn**: job can be preempted, i.e. stopped and started again.
- Many other possibilities: $L$ (lags between jobs), **no-idle** (machines can’t be idle). . .
Job Characteristics, Dependencies

- **prec**: dependencies between jobs. $i \rightarrow i'$ implies if $\sigma(i) = (j, t)$ and $\sigma(i') = (j', t)$, then:

$$t + p_i \leq t'.$$

- In its more general form DAGs (Directed Acyclic Graphs) are considered.

In some cases, a particular class of DAG is given: **tree, forest, chains**...
Objectives

- Completion time: \( C_i = t + p_i \) with \( \sigma(i) = (j, t) \).
- \( C_{\text{max}} \): Minimize **Makespan**, i.e. \( \max_{i \in J} C_i \).
- \( \sum C_i \): Minimize **Flow Time**, i.e. \( \sum_{i \in J} C_i \).
- \( \sum w_i C_i \): Minimize **Weighted Flow Time**, i.e. \( \sum_{i \in J} w_i C_i \).
- \( F_i \): with release date, minimize \( \sum_{i \in J} C_i - r_i \).
- \( L_i \): Minimize **Lateness**, i.e. \( \sum_{i \in J} C_i - d_i \).
- \( T_i \): Minimize **Tardiness**, i.e. \( \sum_{i \in J} \max(0, C_i - d_i) \).
- Other options: Unitary penalty for late jobs, \( L_{\text{max}}, T_{\text{max}} \), non-temporal objectives...
Some examples:

- $P||C_{max}$: minimize makespan for independent jobs on multiple identical machines.
- $1|d_i|\sum L_i$: minimize lateness on a single machine.
- $R2|d_i, tree|\sum T_i$ minimize tardiness on two unrelated machines with tree-shaped dependencies.
- $Q|p_i = 1|\sum w_i C_i$: minimize weighted flow-time with jobs of constant duration 1 on related machines.
All presented problems can be algorithmically solved.

However, fast computation time is needed → we aim for polynomial algorithms if possible (number of operation grows polynomially with $n$ and $m$).

→ No exhaustive algorithms!

In some cases there is optimal solutions that can be polynomially computed.

\begin{align*}
1 || C_{max}, \; 1 || \sum w_i C_i, \; P | p_i = 1 | \sum w_i C_i, \; R || \sum C_i \ldots
\end{align*}
Hardiness of Problems

- Sadly, not always the case!
- Most of scheduling problems are NP-hard! (NP-hard ≃ there is probably no polynomial and optimal algorithm)
  \[ P||C_{\text{max}}, \ 1|\text{prec}||\sum w_i C_i, \ P2||C_{\text{max}}, \ P|p_i = 1, \ \text{tree}|L_{\text{max}} \ldots \]
- Several alternative solutions:
  - Use non-polynomial algorithms (Linear Programming, Branch-and-Bounds algorithms).
  - Use heuristics (non-optimal polynomial algorithms)
  - Use approximation algorithms (non-optimal polynomial algorithms with guarantee performances)
### Study of $1 \mid \sum w_i C_i$

- One machine, $n$ jobs with processing times $p_i$ and weights $w_i$.

<table>
<thead>
<tr>
<th>jobs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_i$</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$p_i$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Let us try some greedy strategies.
Study of $1|| \sum w_i C_i$

- One machine, $n$ jobs with processing times $p_i$ and weights $w_i$.

<table>
<thead>
<tr>
<th>jobs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_i$</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$p_i$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Let us try some greedy strategies: Longest jobs first.

5 × 2 \rightarrow 9 × 1 \rightarrow 12 × 4 \rightarrow 14 × 3 = 109
Study of $1|| \sum w_i C_i$

- One machine, $n$ jobs with processing times $p_i$ and weights $w_i$.

<table>
<thead>
<tr>
<th>jobs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_i$</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$p_i$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Let us try some greedy strategies: Shortest jobs first.
Study of $1 \mid \sum w_i C_i$

- One machine, $n$ jobs with processing times $p_i$ and weights $w_i$.

<table>
<thead>
<tr>
<th>jobs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_i$</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$p_i$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Let us try some greedy strategies: Heaviest jobs first.

$$5 \times 4 \rightarrow 9 \times 3 \rightarrow 11 \times 2 \rightarrow 14 \times 1 = 83$$
Study of $1 || \sum w_i C_i$

One machine, $n$ jobs with processing times $p_i$ and weights $w_i$.

<table>
<thead>
<tr>
<th>jobs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_i$</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$p_i$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Let us try some greedy strategies: Largest $w_i/p_i$ first.

Let the jobs be processed in the order A → C → D → B.

- $A$: $2 \times 2 = 4$
- $C$: $7 \times 4 = 28$
- $D$: $11 \times 3 = 33$
- $B$: $14 \times 1 = 14$

Total = $4 + 28 + 33 + 14 = 79$
One machine, \( n \) jobs with processing times \( p_i \) and weights \( w_i \).

<table>
<thead>
<tr>
<th>jobs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_i )</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>( p_i )</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Let us try some greedy strategies: Largest \( w_i/p_i \) first.
→ Let us prove it is the best algorithm in any case.
Let us consider $\sigma$ optimal and $i, j$ such that $\sigma(j) = \sigma(i) + p_i$ (consecutive jobs) and $w_i/p_i < w_j/p_j$.

Let us consider $\sigma'$ such that $\sigma'(j) = \sigma(i)$, $\sigma'(i) = \sigma'(j) + p_j$ and $\sigma'(k) = \sigma(k)$ otherwise.

If $k \neq i, j$, $C_k^\sigma = C_k^{\sigma'}$.

Thus $\sum w_k C_k^\sigma - \sum w_k C_k^{\sigma'} = w_i C_i^\sigma + w_j C_j^\sigma - (w_i C_i^{\sigma'} + w_j C_j^{\sigma'})$
Optimality of the Algorithm

\[ \sum w_k C_k^\sigma - \sum w_k C_k^{\sigma'} = w_i C_i^\sigma + w_j C_j^\sigma - (w_i C_i^{\sigma'} + w_j C_j^{\sigma'}) \]

\[ = w_i (t + p_i) + w_j (t + p_i + p_j) - w_j (t + p_j) - w_i (t + p_i + p_j) \]

\[ = w_j p_i - w_i p_j \]

\[ \sum w_k C_k^\sigma - \sum w_k C_k^{\sigma'} = \frac{p_i}{w_i} - \frac{p_j}{w_j} > 0 \]

Contradiction with the optimality of \( \sigma \).
Study of $P\|C_{\text{max}}$

- $m$ identical machines, $n$ jobs. Goal: minimize $C_{\text{max}}$.
- NP-hard (even assuming that $m = 2$).

Here a study of an approximation algorithm.
Approximation Algorithm

Definition

An algorithm $A$ is said to be an $\alpha$-approximation for a problem $P$ if for every instance $I$ of $P$,

$$C^I_A \leq \alpha C^I_{opt}$$

with $C^I_A$ value of the solution returned by $A$ with $I$ as input and $C^I_{opt}$ value of the optimal solution for $I$.

- "This algorithm is at worst $\alpha$ times the optimal."
- The closer to 1 is $\alpha$, the better is the approximation.
List Scheduling

Definition

A **List-Scheduling** algorithm is an algorithm which do not allow idle time on machines (if possible).

![Diagram of List-Scheduling algorithm](image)

**Theorem**

Every List-Scheduling algorithm is a $2 - 1/m$-approximation for $P||C_{\text{max}}$. 
List Scheduling

Definition

A List-Scheduling algorithm is an algorithm which do not allow idle time on machines (if possible).

\[ C_{\text{max}} \times r_1 r_2 r_3 r_4 \]

Theorem

Every List-Scheduling algorithm is a \(2 - 1/m\)-approximation for \(P\|C_{\text{max}}\).
Proof of Approximation

- Let $\sigma$ be a schedule from a list-scheduling algorithm for $n$ jobs and $m$ machines.

- $C_{max}^{opt} \geq \frac{\sum_{i=1}^{n} p_i}{m}$ and $\forall i \in J$, $C_{max}^{opt} \geq p_i$.

- Let $k$ be the last job to finish ($C_k = C_{max}^{\sigma}$) and $t$ the time when its execution begins.
Proof of Approximation

Before $t$, all machines are busy: $\sum_{i \neq k} p_i \geq m \times t$.

Thus:

$$t \leq \frac{\sum_{i \neq k} p_i}{m} = \frac{\sum_{i=1}^{n} p_i - p_k}{m} \leq C_{\text{opt max}}^{\sigma} - \frac{p_k}{m}$$

And

$$C_{\text{max}}^{\sigma} = t + p_k \leq C_{\text{max}}^{\text{opt}} + \left(1 - \frac{1}{m}\right)p_k \leq \left(2 - \frac{1}{m}\right)C_{\text{max}}^{\text{opt}}$$
Straightness of the Approximation

- An approximation is said to be straight if the bound is reached.
- The $2 - \frac{1}{m}$ previous bound is straight.

$$C_{\text{max}} = 3$$

$$C_{\text{opt}} = 2$$

$$\frac{C_{\text{max}}}{C_{\text{opt}}} = \frac{3}{2} = 2 - \frac{1}{2}.$$
Exercises

- Show that every list-scheduling is a $2 - \frac{1}{m}$-approximation for $P|\text{prec}|C_{\text{max}}$.
  - First clue: Consider the chain of precedence chain of the last finished job as a lower bound for $C_{\text{opt}}^\text{max}$.
  - Second clue: Assume all jobs that are not in this chain are not executed simultaneously with the ones in the chain.

- Show that LPT (Longest Processing Time first) is a $\frac{4}{3} - \frac{1}{3m}$-approximation for $P||C_{\text{max}}$.
  - First clue: assume the last job to finish is the one with the smallest processing time (denoted $p_{\text{min}}$).
  - Second clue: Show that if $p_{\text{min}} \geq C_{\text{max}}^{\text{opt}}/3$ then $C_{\text{max}} = C_{\text{opt}}^\text{max}$.
  - Third clue: Show the approximation is fulfilled in the remaining case.
Mainly theoretical aspect of scheduling (high level).
"old school scheduling", focus on time constraints and centralized and static decision.
- Energy constraints, Communication constraints.
- Distributed Scheduling.
- dynamic constraints:
  - Instability of platforms.
  - Fault-tolerance.
- Less theoretical approaches also exists!