Contents

- Classical Scheduling
  - Basic Notions
  - Graham Notation
  - Single resource scheduling
  - Parallel resources scheduling
  - Job shop scheduling
Planning vs Scheduling

- **Planning**
  - given a description of the current state, a set of possible actions, and a desired state come up with a sequence of actions = plan that one can take to achieve the desired state.
  - belongs to the category of Artificial Intelligence.
  - high complexity, P-SPACE hard or even Undecidable.

- **Scheduling**
  - given a collection of actions and restricted resources decide how to execute all the actions in an efficient manner (create a schedule).
  - belongs to the category of operations research.
  - complexity typically in P and NP
Scheduling – problem definition

- **Given:**
  - A set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ to be processed
  - A set of machines $\mathcal{M} = \{M_1, \ldots, M_m\}$ to process the jobs
  - Various constraints and properties
    - Interference/dependency of jobs
    - Compatibility of machines and jobs
    - Efficiency of a machine for a given (type of) job
    - Preemptiveness of jobs (can be interrupted or not)
    - ...

- Various Optimization Criteria

- **Task:**
  - Find a **Schedule**, i.e., a mapping of jobs to machines and processing times that satisfies the given constraints and is optimal w.r.t. optimization criteria
Schedule Visualization – Gantt Charts

- (a) machine oriented
- (b) job oriented
Data associated to Jobs

A job $J_j \in \mathcal{J}$ can have a:

- Processing time $p_j$ – time to do the job
- Release date $r_j$ – earliest time when the job can be run
- Due date $d_j$ – called deadline if strict
- Weight $w_j$ – the cost/benefit of doing the job
- Cost function $h_j(t)$ – cost of completing $J_j$ at time $t$
- A job $J_j$ may consist of several ($n_j$) operations (a.k.a. tasks) $J_j \rightarrow O_{j_1}, \ldots, O_{j_{n_j}}$, and data for each operation.
- A set of machines associated to each job/operation
Data associated to Jobs

A job $J_j \in \mathcal{J}$ can have a:

- Processing time $p_j$ – time to do the job
- Release date $r_j$ – earliest time when the job can be run
- Due date $d_j$ – called deadline if strict
- Weight $w_j$ – the cost/benefit of doing the job
- Cost function $h_j(t)$ – cost of completing $J_j$ at time $t$
- A job $J_j$ may consist of several ($n_j$) operations (a.k.a. tasks) $J_j \rightarrow O_{j1}, \ldots, O_{jn_j}$, and data for each operation.
- A set of machines associated to each job/operation

Data that depend on the schedule:

- Starting time $S_j$
- Completion time $C_j$ (typically $C_j = S_j + p_j$)
Graham Notation

A scheduling problem is described by a triplet: $\alpha \big| \beta \big| \gamma$ where

- $\alpha$ describes the machine environment (1-2 entries)
- $\beta$ job characteristics (0-many entries)
- $\gamma$ objective function to be minimized (1 entry)
Graham Notation

A scheduling problem is described by a triplet: $\alpha | \beta | \gamma$ where

- $\alpha$ describes the machine environment (1-2 entries)
- $\beta$ job characteristics (0-many entries)
- $\gamma$ objective function to be minimized (1 entry)

The Scheduling Zoo http://www-desir.lip6.fr/~durrc/query/ is a comprehensive website where you can look up various scheduling problems and their properties.
Objective functions

For a given job $J_j$ it is a function of $C_j$ (completion times) and possibly something extra

- Lateness $L_j = C_j - d_j$ (completion minus due date)
- Tardiness $T_j = \max(L_j, 0)$
- Earliness $E_j = \max(d_j - C_j, 0)$
- Unit Penalty $U_j = T_j > 0 \ ? 1 : 0$
Objective functions

For a given job \( J_j \) it is a function of \( C_j \) (completion times) and possibly something extra

- Lateness \( L_j = C_j - d_j \) (completion minus due date)
- Tardiness \( T_j = \max(L_j, 0) \)
- Earliness \( E_j = \max(d_j - C_j, 0) \)
- Unit Penalty \( U_j = T_j > 0 \) ? 1 : 0

For a schedule

- Makespan \( C_{max} = \max\{C_1, \ldots, C_n\} \)
- Maximum lateness \( L_{max} = \max\{L_1, \ldots, L_n\} \)
- Total completion \( \sum C_i \)
- Total weighted tardiness \( \sum w_i T_i \)
- Weighted number of tardy jobs \( \sum w_i U_i \)
Problem description?
Problem description?

- 1 machine
- job release times are specified
- goal is to minimize maximal lateness (minimize lateness)

This problem is NP hard, but adding further constraints makes it P
Problem description?

### Problem description?

Given a single machine with jobs having specified release times and a common due date, the goal is to minimize the maximum lateness. This problem is known to be solvable in polynomial time (P). How can we achieve this?
1\mid r_j, d_j = d \mid L_{\text{max}}

Problem description?

- 1 machine
- job release times are specified, all jobs have the same due date
- goal is to minimize maximal lateness (minimize lateness)

Solvable in P. How?
$1 | r_j, d_j = d | L_{max}$

Problem description?

- 1 machine
- job release times are specified, all jobs have the same due date
- goal is to minimize maximal lateness (minimize lateness)

Solvable in P. How?

- tasks are ordered in ascending order by release dates
Problem description?

$1 | r_j = r | L_{max}$

Use the earliest due date rule (EDD) – tasks are ordered in ascending order by due dates.
Problem description?

- 1 machine
- job release times are specified and are the same for each job
- goal is to minimize maximal lateness (minimize lateness)

Solvable in P. How?
Problem description?

- 1 machine
- job release times are specified and are the same for each job
- goal is to minimize maximal lateness (minimize lateness)

Solvable in P. How?

- Use the earliest due date rule (EDD) – tasks are ordered in ascending order by due dates
- proof by contradiction
1↓ | \(r_j, pmtn\) | \(L_{max}\)

**Problem description?**

- 1 machine
- job release times are specified, job can be interrupted (preemption)
- goal is to minimize maximal lateness (minimize lateness)

Solvable in P. How?
1\mid r_j,\text{ pmtn}\mid L_{\text{max}}

Problem description?
- 1 machine
- job release times are specified, job can be interrupted (preemption)
- goal is to minimize maximal lateness (minimize lateness)

Solvable in P. How?
- Start with the job $J_j$ with the smallest $r_j$ (break ties by smallest $d_j$)
- as soon as we reach the $r_j$ of a job $J_j$ with smaller $d_j$ than the current jobs due date we interrupt the current job and switch to that job $J_j$

<table>
<thead>
<tr>
<th>task</th>
<th>$p_j$</th>
<th>$r_j$</th>
<th>$\delta_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
$1 | r_j | L_{\text{max}}$

Back to general version (NP Hard)

- 1 machine
- job release times are specified
- goal is to minimize maximal lateness (minimize lateness)

We design a branch and bound algorithm

- Branch on
Back to general version (NP Hard)

- 1 machine
- job release times are specified
- goal is to minimize maximal lateness (minimize lateness)

We design a branch and bound algorithm

- Branch on which job to do next (build schedule left to right)
- Calculate bound by solving a relaxed problem: $1|\text{\it{r}}_j, pmt\text{t}|L_{\text{max}}$
  - An optimal preemptive schedule has always better or equal lateness than the non relaxed problem – it provides a lower bound
  - If we find a schedule that has no interruptions we use that solution, no further search needed

- Pruning: a task $J_j$ is pruned if there is other task $J_i$ that could be completed before $J_j$ can start.
Branch & Bound Example

1. \( r_j | L_{\text{max}} \)

Lower bound is greater than the best so-far solution

Task 2 can be before task 3

Tasks 1 or 2 can be before task 4

<table>
<thead>
<tr>
<th>task</th>
<th>( p_j )</th>
<th>( r_j )</th>
<th>( \delta_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

Final schedule: 1,3,4,2
Planning with Parallel Resources

We have several resources (Machines) to process the tasks. We assume each machine can process each task

- identical resources – each machine has the same speed on each task
- uniform resources – machines have different speed, it does not depend on the task (if $J_1$ can be done $k$ times faster than $J_2$ on one machine, it is $k$ times faster on each machine.)
- general resources – task duration depends on machine arbitrarily

Preemptive tasks may migrate between the machines
Problem description:

- minimize makespan for tasks running on $m$ identical resources

How to do it? (in linear time)
Problem description:

- minimize makespan for tasks running on $m$ identical resources

How to do it? (in linear time)

- A lower bound for makespan $LB = \max\{\max_i p_i, \frac{\sum_i p_i}{m}\}$
- sequence tasks in any order on the first machine, when LB is reached split the last task and schedule on the next machine
- a task will not overlap with itself on another machine because $p_i < LB$
Problem description:

- minimize makespan for non-preemptive tasks with precedence relations between them running on \( m \) identical resources

Complexity?

- if \( 2 \leq m < n \) (more tasks than machines) then NP-hard
- if \( m \geq n \) (more tasks than machines) then P – critical path method
Problem description:
- minimize makespan for non-preemptive tasks with precedence relations between them running on $m$ identical resources

Complexity?
- if $2 \leq m < n$ (more tasks than machines) then NP-hard
- if $m \geq n$ (more tasks than machines) then P – critical path method
Critical Path Method

Terminology:
- critical task – a task that cannot be delayed without increasing the makespan
- critical path – a sequence of critical tasks

Algorithm:
- find the earliest start (est) and completion (ect) time for each task
  - tasks $J_i$ with no predecessors have $est_i = 0$, $ect_i = p_i$
  - a task with predecessors $J_1, \ldots, J_k$ has $est = \max_{i=1}^{k} ect_i$
  - $C_{max} = \max_i ect_i$
- Find the latest start (lst) and completion (lct) time for each task
  - task $J_i$ with no successor $lct_i = C_{max}$ and $lst_i = C_{max} - p_i$
  - a task with successors $J_1, \ldots, J_k$ has $lct = \min_{i=1}^{k} lst_i$
- each task $J_i$ such that $est_i = lst_i$ is a critical task
Shop Problems

- The Shop Problems are the most commonly used scheduling problems in practice
  - Each job consists of a set of tasks
  - Each task must be executed on a specific machine
  - There can be precedence relations between the tasks
- The 3 kinds of shop problems
  - Job-shop – the tasks within each job are totally ordered (a job is sequence of tasks), often each resource is used at most once per job.
  - Flow-shop – special case of Job-shop, all jobs have identical tasks in the same order (assembly line production)
  - Open-shop – no precedence relations between the tasks,

Graham notation: $J_m||C_{max}$, $F_m||C_{max}$, $O_m||C_{max}$ for $m$ machines and optimizing makespan, in general NP-hard, see scheduling zoo polynomial cases and algorithms.
Job Shop by SAT (Crawford Encoding)

Problem Definition:
- we have a set $n$ jobs $J_1, \ldots, J_n$ and $m$ machines $M_1, \ldots, M_m$
- each job $J_i = \langle O_{i1}', \ldots, O_{iq_i}' \rangle$ is a sequence of operations
- each operation $O_{il}'$ requires the exclusive use of machine $M_{O_{il}'}$ for an uninterrupted duration $p_{il}'$ (processing time)

Example: job0 = [(0,3),(1,2),(2,2)], job1=[(0,2),(2,1),(1,4], job2=[(1,4),(2,3)]

For the SAT encoding we will assume that the makespan is at most $L$ (encode the question, is there a schedule with makespan $L$ or less).
Job Shop by SAT – 2

Variables:
- $pr_{i,j}^{l,k}$ means that $O_i^l$ precedes $O_j^k$
- $sa_{i,t}^l$ means that $O_i^l$ starts at time $t$ or after $t$
- $eb_{i,t}^l$ means that $O_i^l$ ends at time $t$ or before $t$

Clauses (1-3):
- $O_i^l$ precedes $O_{i+1}^l$
  - $(pr_{i,i+1}^{l,l})$
Job Shop by SAT – 2

Variables:
- $pr_{i,j}$ means that $O_i$ precedes $O_j$
- $sa_{i,t}$ means that $O_i$ starts at time $t$ or after $t$
- $eb_{i,t}$ means that $O_i$ ends at time $t$ or before $t$

Clauses (1-3):
- $O_i$ precedes $O_{i+1}$
  $$(pr_{i,i+1})$$
- If $O_i$ and $O_j$ require the same machine we add clauses
  $$(pr_{i,j}^{l,k} \lor pr_{j,i}^{l,k})$$
Job Shop by SAT – 2

Variables:
- $pr_{i,j}^{l,k}$ means that $O_i^l$ precedes $O_j^k$
- $sa_{i,t}^l$ means that $O_i^l$ starts at time $t$ or after $t$
- $eb_{i,t}^l$ means that $O_i^l$ ends at time $t$ or before $t$

Clauses (1-3):
- $O_i^l$ precedes $O_{i+1}^l$ 
  ($pr_{i,i+1}^{l,l}$)
- If $O_i^l$ and $O_j^k$ require the same machine we add clauses 
  ($pr_{i,j}^{l,k} \lor pr_{j,i}^{k,l}$)
- If $O_i^l$ starts at $t$ or after $t$ then it also starts after $t - 1$ 
  $sa_{i,t}^l \rightarrow sa_{i,t-1}^l$
Job Shop by SAT – 3

Variables:

- \( pr_{i,j}^{l,k} \) means that \( O_i^l \) precedes \( O_j^k \)
- \( sa_{i,t}^l \) means that \( O_i^l \) starts at time \( t \) or after \( t \)
- \( eb_{i,t}^l \) means that \( O_i^l \) ends at time \( t \) or before \( t \)

Clauses (4-6):

- If \( O_i^l \) end at \( t \) or before \( t \) then it also ends before \( t + 1 \)
  \[ eb_{i,t}^l \rightarrow eb_{i,t+1}^l \]
Job Shop by SAT – 3

Variables:

- $pr_{i,j}^{l,k}$ means that $O_i^l$ precedes $O_j^k$
- $sa_{i,t}^l$ means that $O_i^l$ starts at time $t$ or after $t$
- $eb_{i,t}^l$ means that $O_i^l$ ends at time $t$ or before $t$

Clauses (4-6):

- If $O_i^l$ end at $t$ or before $t$ then it also ends before $t + 1$
  \[ eb_{i,t}^l \rightarrow eb_{i,t+1}^l \]
- If $O_i^l$ starts at $t$ or after $t$ then it cannot end before $t + p_i^l - 1$
  \[ sa_{i,t}^l \rightarrow \neg eb_{i,t+p_i^l-1}^l \]
Job Shop by SAT – 3

Variables:
- $pr_{i,j}^{l,k}$ means that $O_i^l$ precedes $O_j^k$
- $sa_{i,t}^l$ means that $O_i^l$ starts at time $t$ or after $t$
- $eb_{i,t}^l$ means that $O_i^l$ ends at time $t$ or before $t$

Clauses (4-6):
- If $O_i^l$ end at $t$ or before $t$ then it also ends before $t + 1$
  $$eb_{i,t}^l \rightarrow eb_{i,t+1}^l$$
- If $O_i^l$ starts at $t$ or after $t$ then it cannot end before $t + p_i^l - 1$
  $$sa_{i,t}^l \rightarrow \neg eb_{i,t+p_i^l-1}^l$$
- If $O_i^l$ starts at time $t$ or after time $t$ and $O_j^k$ follows $O_i^l$ then $O_j^k$ cannot start before $O_i^l$ is finished
  $$sa_{i,t}^l \land pr_{i,j}^{l,k} \rightarrow \neg sa_{j,t+p_i^l}^k$$