4.1 Interval Scheduling

Interval Scheduling

Interval scheduling.
- Job j starts at s_j and finishes at f_j.
- Two jobs compatible if they don’t overlap.
- Goal: find maximum subset of mutually compatible jobs.

Greedy template. Consider jobs in some order. Take each job provided it’s compatible with the ones already taken.

- [Earliest start time] Consider jobs in ascending order of start time s_j.
- [Earliest finish time] Consider jobs in ascending order of finish time f_j.
- [Shortest interval] Consider jobs in ascending order of interval length f_j - s_j.
- [Fewest conflicts] For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.
Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some order. Take each job provided it’s compatible with the ones already taken.

- breaks earliest start time
- breaks shortest interval
- breaks fewest conflicts

Greedy algorithm. Consider jobs in increasing order of finish time. Take each job provided it’s compatible with the ones already taken.

Sort jobs by finish times so that \( f_1 \leq f_2 \leq \ldots \leq f_n \).

1. jobs selected
2. \( A \leftarrow \emptyset \) to \( n \) {
   1. if (job \( j \) compatible with \( A \))
   2. \( A \leftarrow A \cup \{ j \} \)
3. return \( A \)

Implementation. \( O(n \log n) \).
- Remember job \( j^* \) that was added last to \( A \).
- Job \( j \) is compatible with \( A \) if \( s_j \geq f_j^* \).

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)
- Assume greedy is not optimal, and let’s see what happens.
- Let \( l_1, l_2, \ldots, l_i \) denote set of jobs selected by greedy.
- Let \( J_1, J_2, \ldots, J_m \) denote set of jobs in the optimal solution with \( l_1 = J_1, l_2 = J_2, \ldots, l_i = J_r \) for the largest possible value of \( r \).

Greedy: \( i_3 \) \( i_3 \) \( i_5 \) \( i_{10} \) \( i_{25} \)

OPT: \( J_1 \) \( J_2 \) \( J_3 \) \( J_{10} \) \( \ldots \)

job \( i_{10} \) finishes before \( J_{10} \)

why not replace job \( J_{10} \) with \( i_{10} \)?

Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)
- Assume greedy is not optimal, and let’s see what happens.
- Let \( l_1, l_2, \ldots, l_i \) denote set of jobs selected by greedy.
- Let \( J_1, J_2, \ldots, J_m \) denote set of jobs in the optimal solution with \( l_1 = J_1, l_2 = J_2, \ldots, l_i = J_r \) for the largest possible value of \( r \).

Greedy: \( i_3 \) \( i_3 \) \( i_5 \) \( i_{10} \) \( i_{25} \)

OPT: \( J_1 \) \( J_2 \) \( J_3 \) \( J_{10} \) \( \ldots \)

job \( i_{10} \) finishes before \( J_{10} \)

why not replace job \( J_{10} \) with \( i_{10} \)?

solution still feasible and optimal, but contradicts maximality of \( r \).
4.1 Interval Partitioning

Interval partitioning.
- Lecture \( j \) starts at \( s_j \) and finishes at \( f_j \).
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses 4 classrooms to schedule 10 lectures.

Def. The depth of a set of open intervals is the maximum number that contain any given time.

Key observation. Number of classrooms needed \( \geq \) depth.

Ex: Depth of schedule below = 3 \( \Rightarrow \) schedule below is optimal.

Q. Does there always exist a schedule equal to depth of intervals?

Intervals Partitioning: Lower Bound on Optimal Solution
4.2 Scheduling to Minimize Lateness

Interval Partitioning: Greedy Algorithm

**Greedy algorithm.** Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

Sort intervals by starting time so that \( s_1 \leq s_2 \leq \ldots \leq s_n \). Let \( d = 0 \) be the number of allocated classrooms.

for \( j = 1 \) to \( n \) {
  if (lecture \( j \) is compatible with some classroom \( k \))
    schedule lecture \( j \) in classroom \( k \)
  else
    allocate a new classroom \( d + 1 \)
    schedule lecture \( j \) in classroom \( d + 1 \)
    \( d \leftarrow d + 1 \)}

Implementation. \( O(n \log n) \).

- For each classroom \( k \), maintain the finish time of the last job added.
- Keep the classrooms in a priority queue.

Interval Partitioning: Greedy Analysis

**Observation.** Greedy algorithm never schedules two incompatible lectures in the same classroom.

**Theorem.** Greedy algorithm is optimal.

**Pf.**

- Let \( d \) be the number of classrooms that the greedy algorithm allocates.
- Classroom \( d \) is opened because we needed to schedule a job, say \( j \), that is incompatible with all \( d-1 \) other classrooms.
- Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than \( s_j \).
- Thus, we have \( d \) lectures overlapping at time \( s_j + \epsilon \).
- Key observation: all schedules use \( \geq d \) classrooms.

Scheduling to Minimizing Lateness

**Minimizing lateness problem.**

- Single resource processes one job at a time.
- Job \( j \) requires \( t_j \) units of processing time and is due at time \( d_j \).
- If \( j \) starts at time \( s_j \), it finishes at time \( f_j = s_j + t_j \).
- Lateness: \( \ell_j = \max \{ 0, f_j - d_j \} \).
- Goal: schedule all jobs to minimize maximum lateness \( L = \max \ell_j \).

**Ex:**

<table>
<thead>
<tr>
<th>( t_j )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_j )</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( d_1 = 9 )</th>
<th>( d_2 = 8 )</th>
<th>( d_3 = 15 )</th>
<th>( d_4 = 6 )</th>
<th>( d_5 = 14 )</th>
<th>( d_6 = 9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>lateness = 2</td>
<td>lateness = 0</td>
<td>max lateness = 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$.
- [Earliest deadline first] Consider jobs in ascending order of deadline $d_j$.
- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

```
Sort n jobs by deadline so that $d_1 \leq d_2 \leq \ldots \leq d_n$.
t ← 0
for j = 1 to n
    Assign job j to interval [t, t + t_j]
    $s_j ← t$, $f_j ← t + t_j$
    t ← t + t_j
output intervals [s_j, f_j]
```

Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t_j</td>
<td>10</td>
</tr>
<tr>
<td>d_j</td>
<td>100</td>
</tr>
</tbody>
</table>

counterexample

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t_j</td>
<td>10</td>
</tr>
<tr>
<td>d_j</td>
<td>2</td>
</tr>
</tbody>
</table>

counterexample

Minimizing Lateness: No Idle Time

Observation. There exists an optimal schedule with no idle time.

- d = 4

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Observation. The greedy schedule has no idle time.
Minimizing Lateness: Inversions

**Def.** An inversion in schedule $S$ is a pair of jobs $i$ and $j$ such that: $i < j$ but $j$ scheduled before $i$.

**Observation.** Greedy schedule has no inversions.

**Observation.** If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

Minimizing Lateness: Analysis of Greedy Algorithm

**Theorem.** Greedy schedule $S$ is optimal.

**Pf.** Define $S^*$ to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume $S^*$ has no idle time.
- If $S^*$ has no inversions, then $S = S^*$.
- If $S^*$ has an inversion, let $i$-$j$ be an adjacent inversion.
  - swapping $i$ and $j$ does not increase the maximum lateness and strictly decreases the number of inversions
  - this contradicts definition of $S^*$.

Greedy Analysis Strategies

**Greedy algorithm stays ahead.** Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

**Exchange argument.** Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

**Structural.** Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

\[
\begin{align*}
\text{if job } j \text{ is late:} \\
L_i & = f_i - d_i \\
L_j & = f_j - d_j \\
\text{let } S^* \text{ have no inversions, then } S = S^*. \\
\text{if } j \text{ finishes at time } f_j, \\
\text{if } j < i, \\
\text{the lateness before the swap, and let } L_i' \text{ be it afterwards,} \\
\text{for all } k \neq i, j, \\
L_j' & = L_j - d_j \\
L_i' & = L_i \\
\text{if job } j \text{ is late:} \\
L_j' & = f_j - d_j \\
L_i' & = f_i - d_i \\
L_i & = L_i \\
\end{align*}
\]