CMSC 451: Minimizing Lateness

Slides By: Carl Kingsford

Department of Computer Science
University of Maryland, College Park

Based on Section 4.3 of Algorithm Design by Kleinberg & Tardos.
Minimizing Lateness

A new kind of scheduling problem. Not given intervals with start and end times.

Instead, requests are of the form \((t_i, d_i)\), where:

- \(t_i\) is the job length, and
- \(d_i\) is the job deadline

We want to schedule these jobs on a single processor.

**Definition**

The lateness \(L_i\) of job \(i\) is \(\max\{0, f_i - d_i\}\), i.e. the length of time past its deadline that it finishes.

Our goal: minimize the maximum lateness.
Example
Possible Greedy Rules

- Short jobs first.
Possible Greedy Rules

- Short jobs first.

```
<table>
<thead>
<tr>
<th>t_1 = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------</td>
</tr>
<tr>
<td>t_2 = 100</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>d_2 = 100</td>
</tr>
</tbody>
</table>
```
Possible Greedy Rules

- Short jobs first.

- Smallest slack time: $d_i - t_i$. 

\[
\begin{align*}
  t_1 &= 10 \\
  t_2 &= 100 \\
  d_2 &= 100 \\
  d_1 &= 110
\end{align*}
\]
Possible Greedy Rules

- Short jobs first.

- Smallest slack time: $d_i - t_i$.
• **Earliest deadline first**: get the job with the most pressing deadline done first.

Surprisingly, don’t need to consider the length of the job!

The algorithm:

Let $d[1] \leq \ldots \leq d[n]$ be the jobs sorted by increasing deadline

Let $f = s$

For $i = 1, \ldots, n$:

Schedule job $i$ starting from time $f$ to $f + t[i]$

Let $f = f + t[i]$
Proving Correctness

Idle time = gaps in the schedule.

Lemma

There is an optimal schedule with no idle time.

Note: some optimal solutions may have idle time.

Given an OPT schedule with gaps, closing the gaps can only decrease the maximum lateness.
The Exchange Argument

- There may be lots of optimal schedules.
- Let $A$ be the schedule produced by our algorithm.
- We’ll start with some optimal solution $OPT$.
- We make local changes to $OPT$, trying to transform it into $A$.
- Each local change will preserve optimality.
Inversions

Definition

An inversion is a pair of jobs $i, j$ such that $i$ is scheduled before $j$, but $d_j < d_i$. 

inversion $(i,j)$
Inversions in Schedules

Lemma

All schedules with no inversions and no idle time have the same maximum lateness.

If there are no inversions, this means the jobs are sorted by increasing deadlines. Let $S_1$ and $S_2$ be two such schedules.

The only way these could differ is when several jobs $i_1, \ldots, i_k$ have the same deadline, which must be adjacent in the schedule.

The last of these jobs is the only one that matters for maximum lateness.

And it doesn’t matter what the order of these jobs are.
Proof Outline

Property P
A schedule has Property P if it has no idle time & no inversions.

1. Our solution has property P
2. All solutions with property P have the same lateness
3. An optimal solution has property P
Proof Outline

Property P
A schedule has Property P if it has no idle time & no inversions.

1. Our solution has property P
   True by construction.

2. All solutions with property P have the same lateness
   True by last lemma

3. An optimal solution has property P
Proof of (3), outline

Theorem

There is an optimal solution with no idle time & no inversions.

- **Idea:** Start with any optimal solution $OPT_0$.
- Flip two adjacent jobs that are an inversion.
- This will reduce the number of inversions without increasing the maximum lateness.
- Repeat until we have no inversions.
(a) If $\text{OPT}_0$ has an inversion, it must have an inversion $i, j$ where $i$ and $j$ are adjacent:

If $a, b$ is an inversion, then $d_a > d_b$, so there must be some step down.

(b) If we swap $i$ and $j$ we reduce the number of inversions by 1.
(c) Let \( OPT_1 \) be the schedule with \( i \) and \( j \) swapped. \( OPT_1 \) has maximum lateness \( \leq OPT_0 \).
Summary for Minimizing Maximum Lateness

So: If we keep swapping adjacent inversions in an optimum solution, we will eventually arrive a solution with no inversions without changing the maximum lateness.

Our greedy algorithm produced a solution without inversions.

Since all solutions without inversions have the same maximum lateness, our greedy algorithm (sort by deadline) must have the same maximum lateness as the optimum.