Min Lateness Scheduling (KT 4.2)

Frank

September 19, 2021

1 Problem

1.1 Input

- n jobs that need to be done $(1, \ldots, n)$
- release time r (schedule from now)
- for each job i, we have $t(i)$ length of job i, and $d(i)$, the deadline of job i

1.2 Intuition

Jobs feeded into a single machine which schedules jobs to be performed one at a time. We want to *miss the deadline by as little as possible*, but of course ideally before deadline.

Order the jobs such that we *minimize* the *maximum amount by which any job misses its deadline*

1.3 Terminology/Notation

- Schedule s is a function that maps a job i to the start time of job i
  - $s(i) \geq r$ is the start time for job i
  - $s(i) + t(i)$ is the finish time for job i
- Note that we have only 1 machine, so intervals of jobs performed should be disjoint. Formally, $[s(i), s(i) + t(i)]$ and $[s(j), s(j) + t(j)]$ do not overlap $\forall i \neq j$
- Lateness of job i in $s = l(i, s) = max(((s(i) + t(i)) - d(i), 0)$, where $s(i) + t(i)$ is the finish time of job i, and lateness of 0 means a job did not miss its deadline.
- Lateness of entire schedule $s = L(s) = max(l(i, s)), 1 \leq i \leq n$ or maximum lateness of any job in s

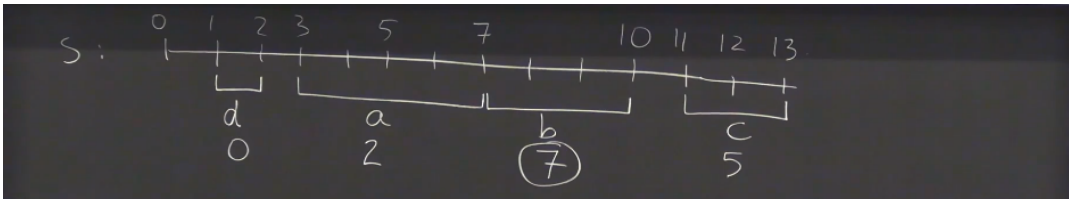
1.4 Output

- A schedule with minimum lateness.

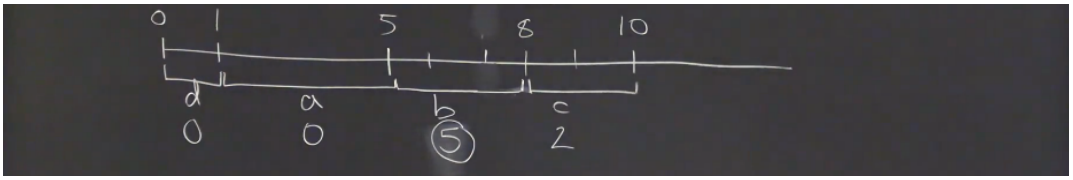
2 Examples

Take input set of jobs $n = 4, r \geq 0$

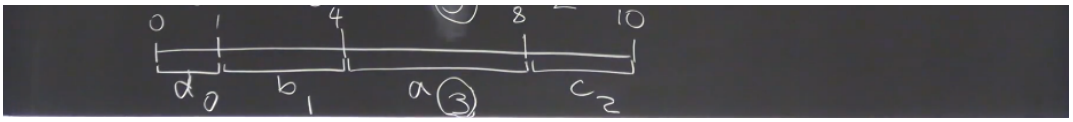
	t-length	d - deadline
a	4	5
b	3	3
c	2	8
d	1	2



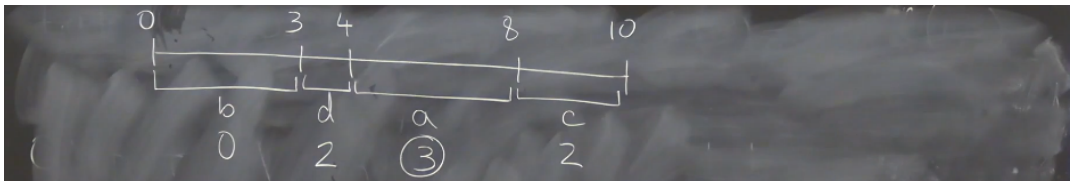
- Wasteful schedule (gaps)
- $L(s) = 7$



- No gaps but still not optimal
- $L(s) = 5$



- Optimal, but not uniquely optimal
- $L(s) = 3$
- Smaller combined lateness compared with the next example, but that's not what we are optimizing



- Another optimal schedule

3 Algorithm

```
1: procedure GREEDY
2:     sort jobs by non-decreasing deadline (lowest deadline or greatest urgency first)
3:     d[1, ..., n] ← deadlines of jobs (sorted)
4:     t[1, ..., n] ← corresponding lengths of jobs (not necessarily sorted)
5:     F ← r
6:     for (i ... n) do
7:         gs(i) ← F
8:         F ← gs(i) + t(i)                          ▷ Schedule gs → greedy start
9:     return gs
```

3.1 Running Time

$O(n \log n) + O(n) = O(n \log n)$

4 Correctness

- Define **inversion** in s: pair of jobs $i\&j$ such that $s(i) < s(j)$ but $d(i) > d(j)$
  Which contradicts the algorithm
- Note: our algorithm creates schedule with no gaps & no inversions
- Since maximum number of inversions when sorted by non-increasing deadline is $O(n^2)$, we have number of inversions ranges from 0 to n^2

5 Proof

5.1 Claim 1:

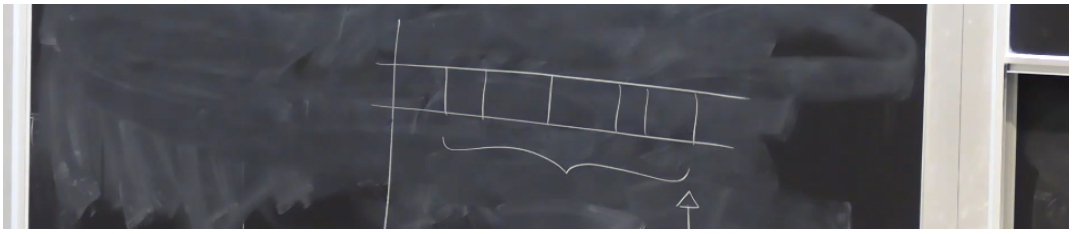
$\exists$ optimal schedule with no gaps.

- Since if gaps exist, we only have to remove the gap, and the schedule should still be optimal
- $\therefore$ only look at schedule with no gaps
- Note that a brute force approach could be to look at every possible combination of intervals to determine the optimal one ($O(n!)$)

5.2 Claim 2:

All schedules with no gaps & no inversions have the same (maximum) lateness

- If 2 different schedules both have no gaps nor inversions, then there must be 2 jobs that have the same deadline but are ordered differently. (Scheduled differently)



5.3 Theorem

Algorithm produces an optimal schedule (minimizes max lateness).

- Strategy: Prove that $\exists$ optimal schedule with no gaps nor inversions, because all schedules with no gaps nor inversions have the same lateness, and thus if one schedule is optimal, then it follows that all schedules with no gaps nor inversions are optimal
- Note that our algorithms produces a schedule with no gaps nor inversions.

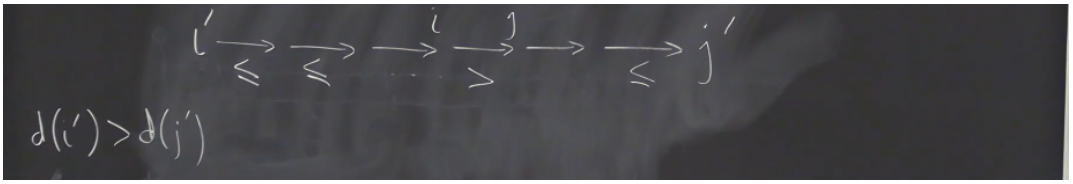
5.4 Proof

By claims 1 & 2, it is enough to show that $\exists$ optimal schedule with no gaps nor inversions.

Let s be any optimal solution with no gaps (claim 1)

If s has no inversions, we're done

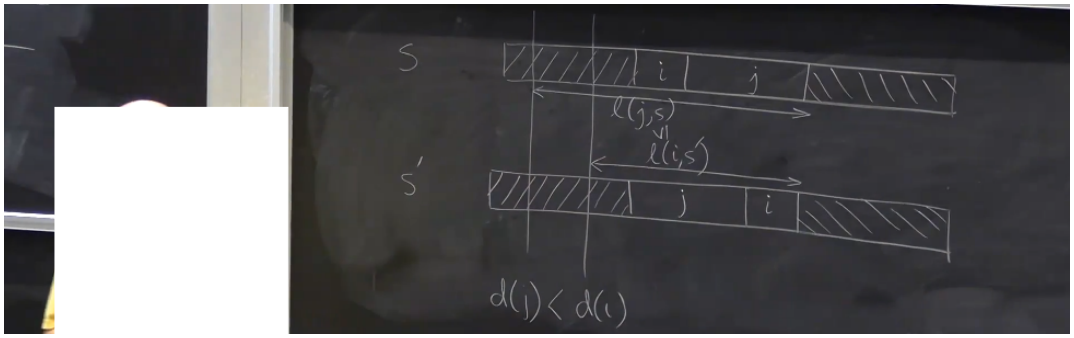
So assume s has some inversions, say between $i'\&j'$, then there is an inversion between 2 successive ($s(j') = f(i')$) jobs in s, meaning $d(i') > d(j')$, where j' is the successor to i'



For the above figure

> if $d(i') > d$(successor of i') then we're done
>
> If not, we walk from i' to j and eventually reach a pair that is inverted, else we contradict the fact that $d(i') > d(j')$ (all successors of i' have deadlines greater than i'). Take that pair to be i and j which are inverted.

In an unrelated note, if we are walking on black or white tiles and we go from a white tile to a black tile, then at some point, we must have crossed from a white tile to a black tile. The above problem is analogous



take a schedule s with 2 jobs $i\&j$ and there is an inversion between the 2

create s' just like s, but swap i and j to fix the inversion.

Facts:

- number of inversions in $s' <$ number of inversions in s
- $l(k, s') = l(k, s)\ \forall k \neq (i \cup j)$
- $l(j, s') \leq l(j, s)$ (lateness of j may have decreased)
- $l(i, s') \geq l(i, s)$ (lateness of i may have increased) (mitigated)

  but there is an inversion between i and j in s
  $\implies d(j) < d(i)$
  Note $l(i, s')$ then see that $l(j, s)$ is at least as big as $l(i, s')$
  $\therefore$ there is a job in s whose lateness is at least as big as $l(i, s')$, and thus $l(i, s')$ will not be the max lateness
  $\therefore$ max lateness of any job in $s' \leq$ max lateness of any job in s
  $\therefore l(i, s') \leq l(j, s) \implies l(i, s') \leq l(j, s') \leq l(j, s)$ or that the lateness of j in s will be the greatest and guarantees that we have a schedule s' with fewer inversions and no greater lateness than s

$\therefore$ we have a schedule with fewer inversions and no greater lateness. Repeat to end up with a schedule that has no inversions but is optimal, since we started with an optimal schedule and guaranteed that subsequent schedules will also be optimal.

$L(s') \leq L(s) \implies L(s') = L(s)$

$\therefore$ we have an optimal schedule with no gaps nor inversions, which proves that our algorithm produces an optimal solution $\square$