Scheduling to Minimize Lateness

Chap 4 Tardos

10-02-2022

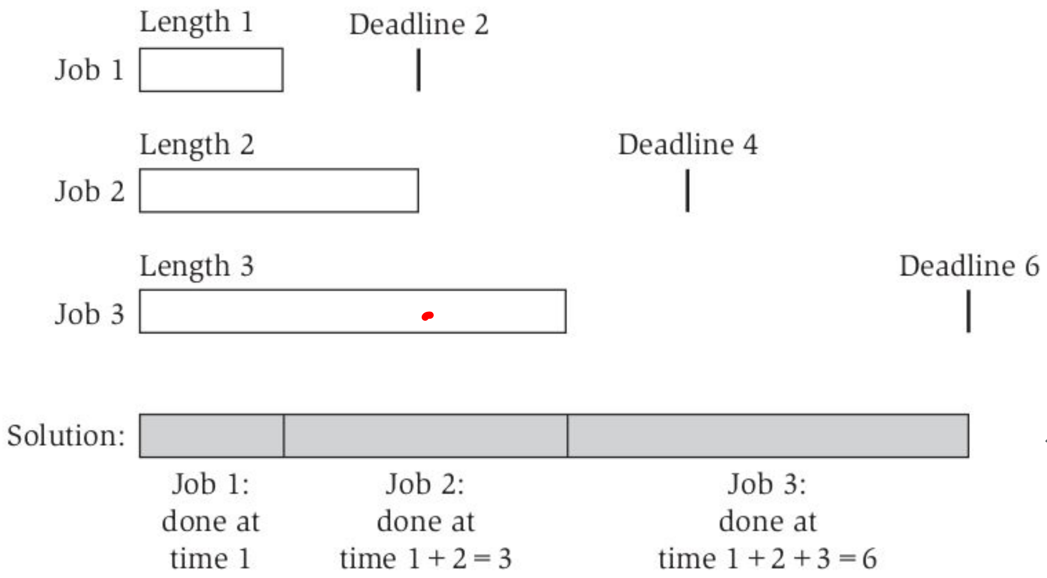
$\longrightarrow$ Variant of Interval Scheduling

Problem

- **Same as before:** n jobs; 1 resource available from some start time s; $\leftarrow$
- **Different from before:** no start-time, end-time, only deadlines (d_i) and time-interval (t_i)
- Jobs willing to be scheduled anytime before the deadline
- Assigned job will require time interval of length t_i
- No two jobs can be allotted the same time-interval. Why?
- So, if a job starts at s_i, it must end at $f_i = s_i + t_i$
- A job is considered late if $f_i > d_i$

Goal

To schedule all jobs and some jobs can be late.

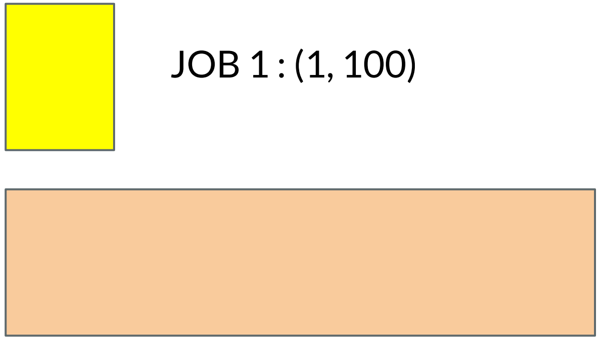


Incurs a maximum lateness of 0.

Algorithm

- **Input data:** pairs of the form (t_i, d_i)
- **Approach 1:** schedule the jobs in order of increasing length t_i , so as to get the short jobs out of the way quickly.
- Do you think it is optimal?

Counter - example



JOB 1 : (1, 100)

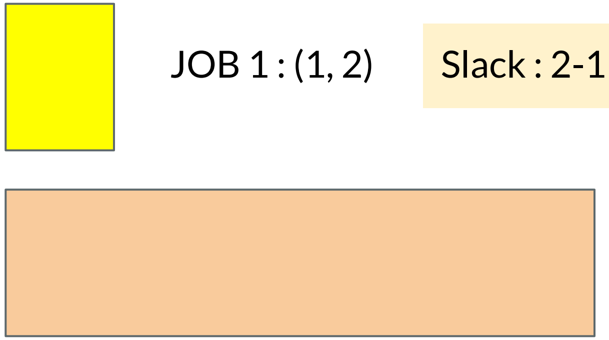
JOB 2: (10, 10)

Our approach: Schedule the job with the shorted running time first -> ignore the deadline?
Our solution: max-lateness = 1
Optimal solution: max-lateness = 0

Algorithm

- **Input data:** pairs of the form (t_i, d_i)
- Approach 1 failed because the job whose slack time $(d_i - t_i)$ is quite small were not into consideration.
- **Approach 2:** schedule jobs in the increasing order of slack time $(d_i - t_i)$
- Do you think it is optimal?

Counter example



JOB 1 : (1, 2)

Slack : 2-1 = 1

JOB 2: (10, 10)

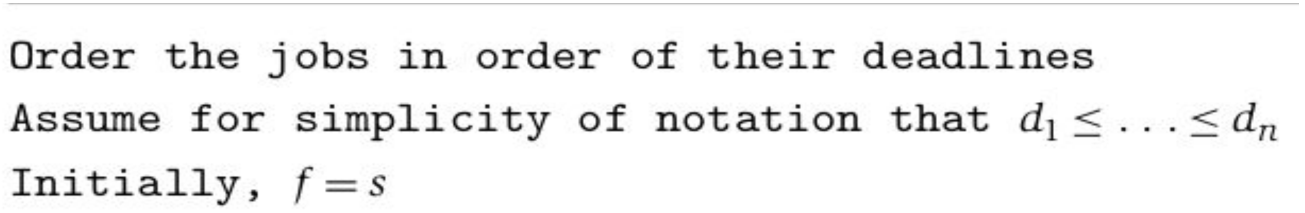
Slack: 10-10 = 0

Our approach: schedule jobs in the increasing order of slack time (di-ti)
Our solution: max-lateness = 9
Optimal solution: max-lateness = 1

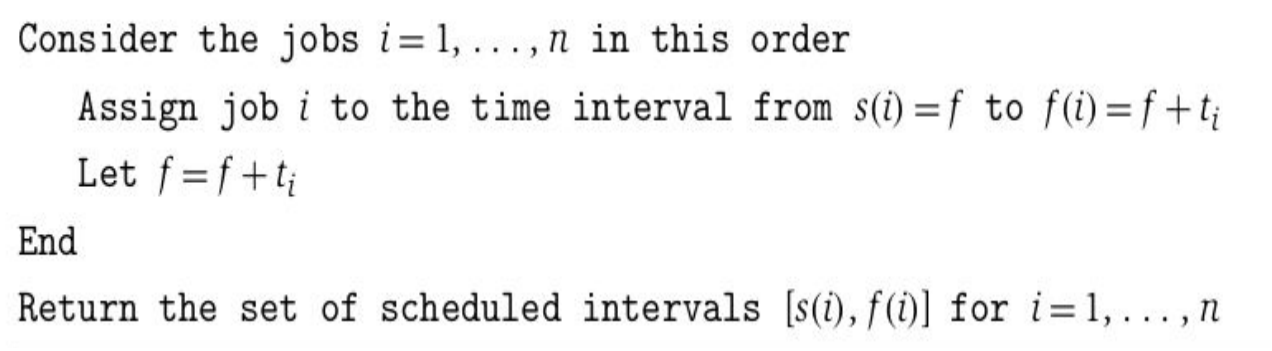
Algorithm

- **Input data:** pairs of the form (t_i, d_i)
- Approach 1 failed because the job whose slack time (d_i-t_i) is quite small were not into consideration.
- Approach 2 failed because the slack time gives the measure of how earlier a job is to be scheduled but forgets about the deadlines
- **Approach 3:** schedule jobs in increasing order of deadlines (Earliest Deadline First) -> ignore the time-intervals?
- Do you think it is optimal?

Algorithm



```
Order the jobs in order of their deadlines
Assume for simplicity of notation that $d_1 \leq \ldots \leq d_n$
Initially, $f = s$
```



```
Consider the jobs $i = 1, \ldots, n$ in this order
    Assign job $i$ to the time interval from $s(i) = f$ to $f(i) = f + t_i$
    Let $f = f + t_i$
End
Return the set of scheduled intervals $[s(i), f(i)]$ for $i = 1, \ldots, n$
```

There is an optimal schedule with no idle time.

How do you prove it?

- Using exchange argument.
- Start by considering an optimal schedule O.
- Gradually modify O, preserving its optimality at each step, but eventually transforming it into a schedule that is identical to the schedule A found by the greedy algorithm.
- We refer to this type of analysis as an **exchange argument.**
- It is a powerful way to think about greedy algorithms in general.

Inversions in this algorithm

- We say that a schedule A' has an inversion if a job i with deadline d_i is scheduled before another job j with earlier deadline $d_j < d_i$.
- Will our greedy algorithm have inversions?
- Will our greedy algorithm have inversions if all d_i's are equal?

Inversions in this algorithm

- We say that a schedule A' has an inversion if a job i with deadline d_i is scheduled before another job j with earlier deadline $d_j < d_i$.
- Will our greedy algorithm have inversions?
- Will our greedy algorithm have inversions if all d_i's are equal?

Our algorithm displays this property ⤵

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

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

If two different schedules A and A’ have neither inversions nor idle time, then they might not produce exactly the same order of jobs, but they can only differ in the order in which jobs with identical deadlines are scheduled.

Does the order of jobs having identical deadlines change the optimal schedule?

Consider such a deadline d. In both schedules A and A’, the jobs with deadline d are all scheduled consecutively (after all jobs with earlier deadlines and before all jobs with later deadlines).

Among the jobs with deadline d, the last one has the greatest lateness, and this lateness does not depend on the order of the jobs. Hence, proved.

Main step

- Now we have characterised the algorithm: no inversions, no idle-time
- The main step in showing the optimality of our algorithm is to establish that there is an optimal schedule that has no inversions and no idle time. To do this, we will start with any optimal schedule having no idle time; we will then convert it into a schedule with no inversions without increasing its maximum lateness. Thus the resulting scheduling after this conversion will be optimal as well.

To prove: There is an optimal schedule that has no inversions and no idle time.

(a) **If O has an inversion, then there is a pair of jobs i and j such that j is scheduled immediately after i and has $d_j < d_i$.** (Why immediately?)

Now suppose O has at least one inversion, and by (a), let i and j be a pair of inverted requests that are consecutive in the scheduled order.

We will decrease the number of inversions in O by swapping the requests i and j in the schedule O.

The pair (i, j) formed an inversion in O, this inversion is eliminated by the swap, and no new inversions are created. Thus we have

(b) **After swapping i and j we get a schedule with one less inversion.**

Is the inverted schedule also optimal?

- The initial schedule O can have at most ${}^nC_2$ inversions (if all pairs are inverted), and hence after at most ${}^nC_2$ swaps we get an optimal schedule with no inversions.

Alert!! New variables are invented here.

- Consider schedule O and Ō. O is the one with inversions. Ō is the one after swapping inverted jobs.
- O: assume that each request r is scheduled for the time interval [s(r), f (r)] and has lateness l'_r . Let $L' = \max_r l'_r$ denote the maximum lateness of this schedule.
- Ō: assume that each request r is scheduled for the time interval $[\bar{s}(r), \bar{f}(r)]$ and has lateness $\bar{l}_r$. Let $\bar{L} = \max_r \bar{l}_r$ denote the maximum lateness of this schedule.

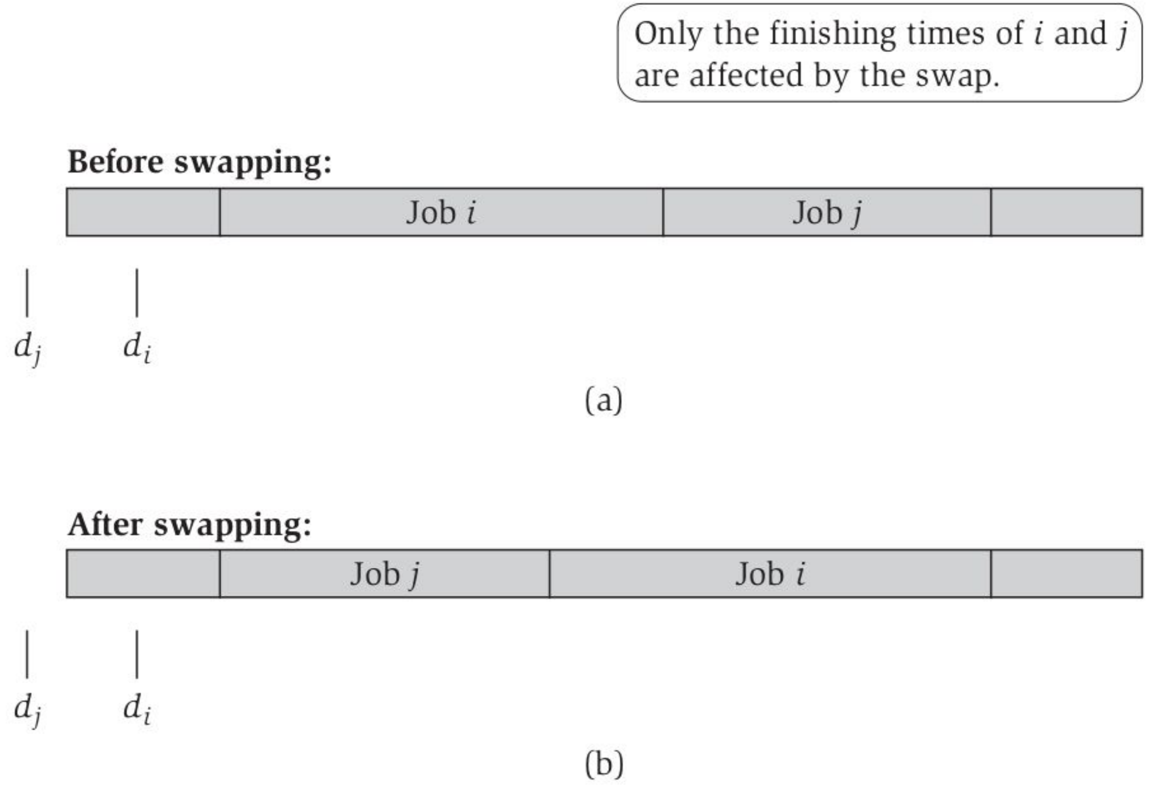


Figure 4.6 The effect of swapping two consecutive, inverted jobs.

- The finishing time of j before the swap is exactly equal to the finishing time of i after the swap.
- Thus all jobs other than jobs i and j finish at the same time in the two schedules.
- Moreover, job j will get finished earlier in the new schedule, and hence the swap does not increase the lateness of job j.
- What about lateness of i? Has it increased?

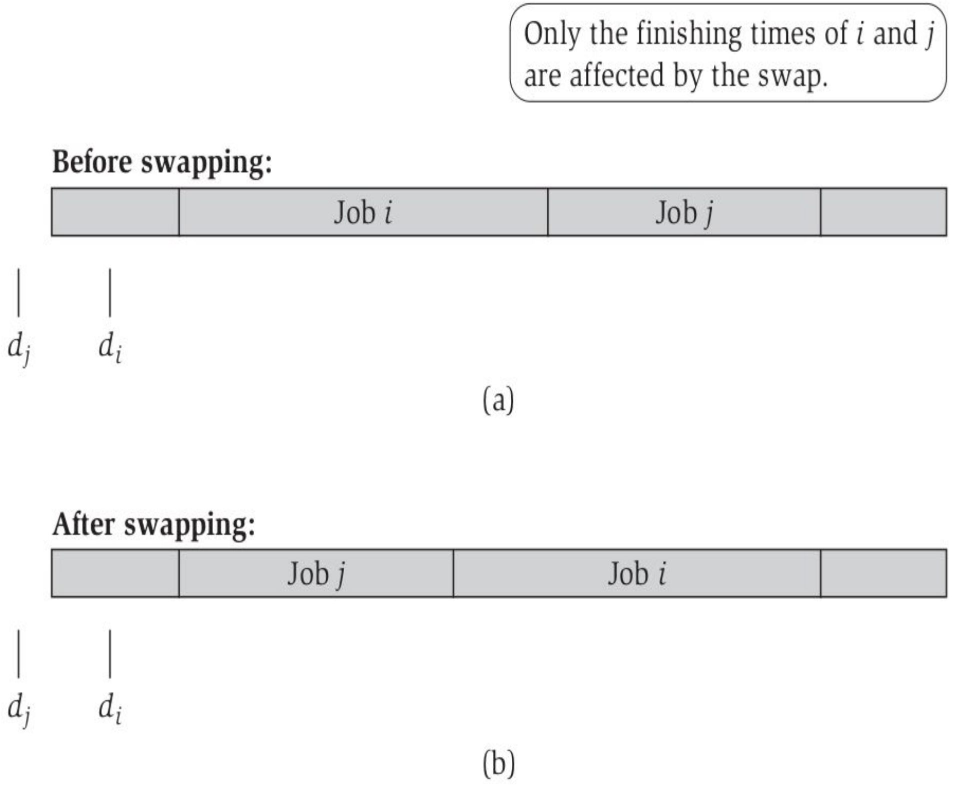


Figure 4.6 The effect of swapping two consecutive, inverted jobs.

- After the swap, job i finishes at time f (j)
- Lateness of i: is $\bar{l}_i = \bar{f}(i) - d_i = f(j) - d_i$.
- i cannot be more late in the schedule $\bar{O}$ than j was in the schedule O.
- $d_i > d_j$ implies that $\bar{l}_i = f(j) - d_i < f(j) - d_j = l'_j$.
- Since the lateness of the schedule O was $L' \geq l'_j > \bar{l}_i$, this shows that the swap does not increase the maximum lateness of the schedule.

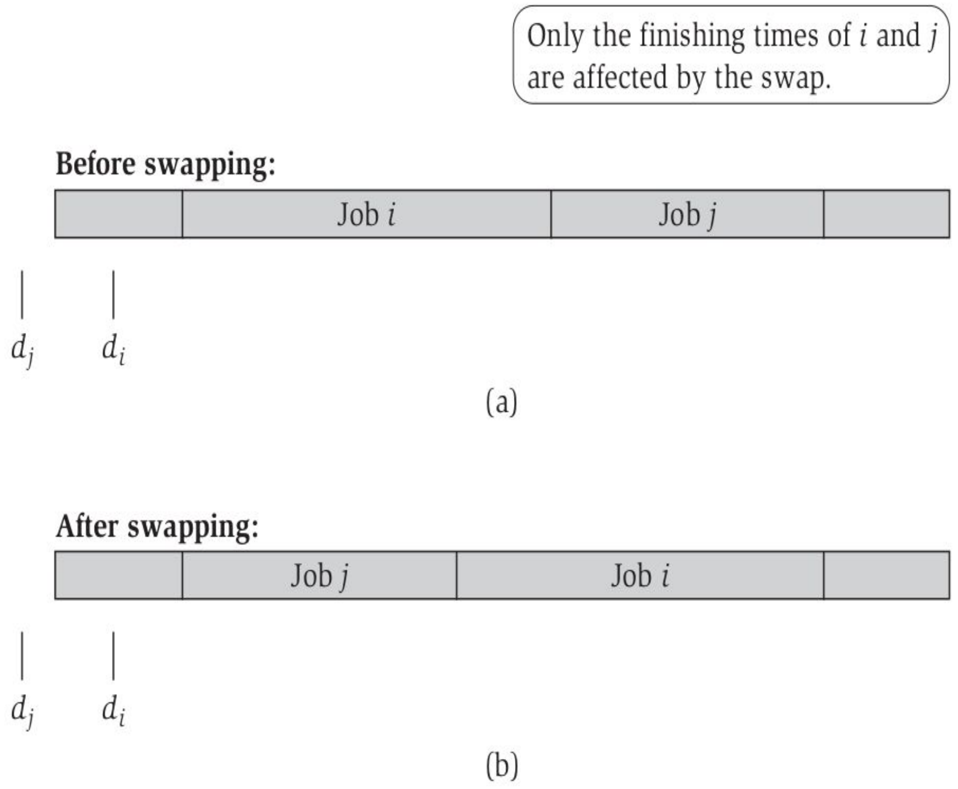


Figure 4.6 The effect of swapping two consecutive, inverted jobs.

Lab exercises

① Interval partitioning

② Scheduling to minimise lateness

→ Use C++ to implement

→ Use files for input

→ Both problems should produce optimal solutions and optimal values as outputs.

Due: 13th February 2022 , 11:59 PM