CS 7800: Advanced Algorithms

Class 3: Greedy Algorithms II

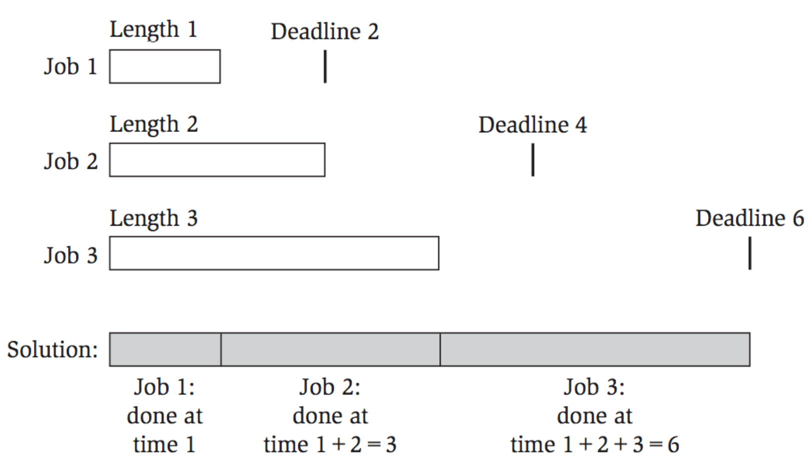
- Finish Minimum Lateness Scheduling
- Minimum Spanning Tree

Jonathan Ullman

September 9, 2025

Minimum Lateness Scheduling

- Input: n jobs with length t_i and deadline d_i
  - Simplifying assumption: all deadlines are distinct
- Output: a minimum-lateness schedule for the jobs
  - Job i starts at s_i finishes f_i, no jobs overlap
  - The lateness of job i is $\max\{f_i - d_i, 0\}$
  - The lateness of a schedule is $\max_i\{\max\{f_i - d_i, 0\}\}$



Enough to determine the best order for the jobs

Possible Greedy Rules

- Do longest jobs first ($t_1 \geq t_2 \geq \dots \geq t_n$)

job 1 ($t=100$, $d=200$)

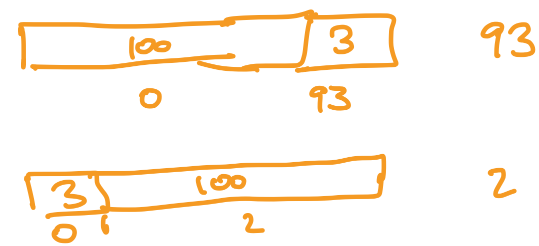
job 2 ($t=3$, $d=10$)

job 3 ($t=3$, $d=10$)

- Do "tightest" deadline first ($d_1 - t_1 \leq d_2 - t_2 \leq \dots \leq d_n - t_n$)

job 1 ($t=100$, $d=101$)

job 2 ($t=3$, $d=10$)



- Do earliest deadline first ($d_1 \leq d_2 \dots \leq d_n$)

We will prove that this rule works

Greedy Algorithm: Earliest Deadline First

- Sort jobs so that $d_1 \leq d_2 \leq \cdots \leq d_n$
- For i = 1, … , n:
  - Schedule job i right after job $i - 1$ finishes

Thm: This algorithm outputs a min-lateness schedule.

Proof: $\boxed{\text{Exchange Argument}}$

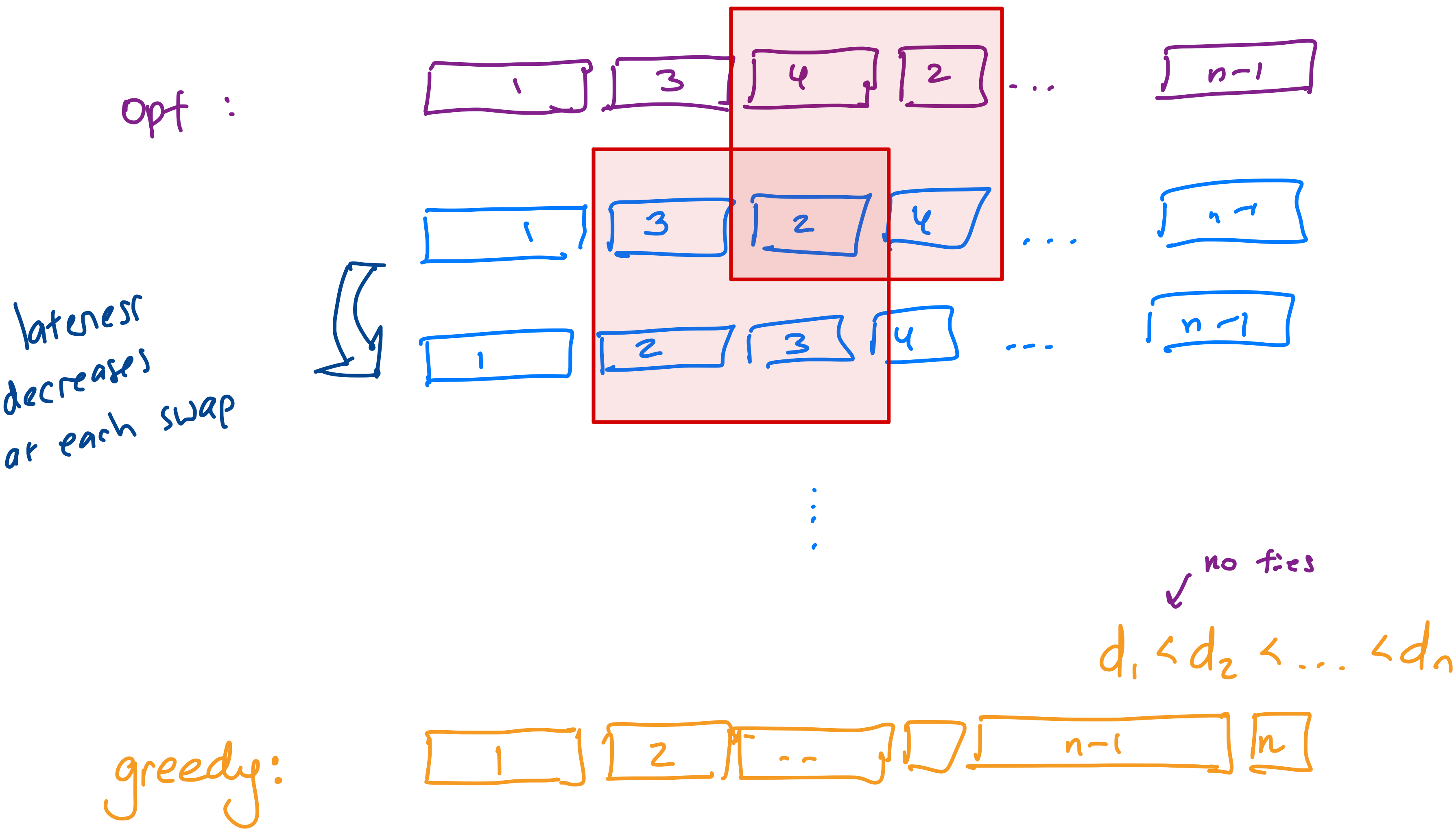
Exchange Argument

① Prove that it is "sufficient" for the algorithm to work for $n=2$

② Prove that the algorithm works for $n=2$

Exchange Argument

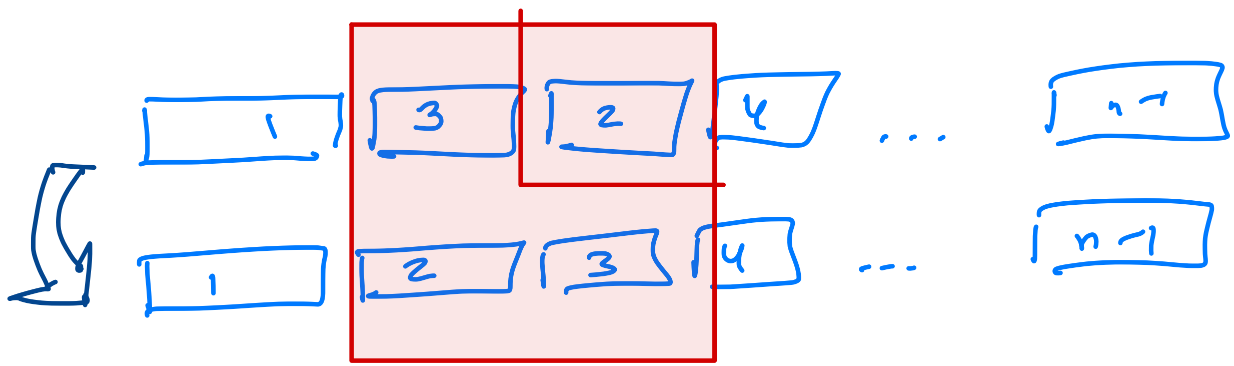
Want to show $\boxed{\text{lateness of greedy} \leq \text{lateness of opt}}$



Exchange Argument

- Can transform opt to greedy by a sequence where each step in the sequence is swapping to consecutive jobs into the right order

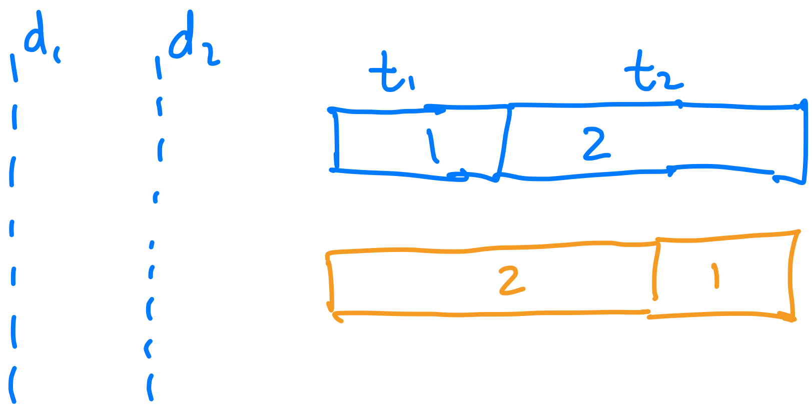
lateness decreases at each swap



Enough to show that sorting two consecutive jobs by deadline never makes lateness worse

Exchange Argument

<u>Claim:</u> For $n=2$ jobs, scheduling $d_1 \le d_2$ is optimal



lateness $1 \to 2$: $\max\{t_1 - d_1, t_1 + t_2 - d_2\}$

lateness $2 \to 1$: $t_1 + t_2 - d_1$

$$t_1 + t_2 - d_1 \geq t_1 - d_1$$

$$t_1 + t_2 - d_1 \geq t_1 + t_2 - d_2$$

$$t_1 + t_2 - d_1 \geq \max\{t_1 + t_2 - d_2, t_1 - d_1\}$$

Exchange Argument

Exchange Argument

Exchange Argument

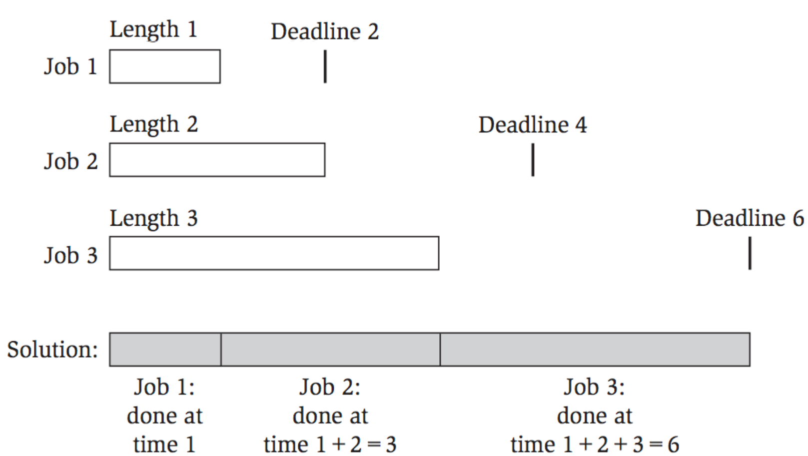
- **Putting the steps together (a thought experiment)**
  - (1) The greedy schedule G has no inversions
  - (2) While O is **not** equal to G
    - (2a) O has at least one inversion
    - (2b) O has a pair of consecutive jobs i, j that are inverted
    - (2c) Swap the order of i, j to fix the inversion
  - (3) Now O is equal to G but its lateness didn't increase, so O started at least as late as G

specific to each problem

can't increase lateness

Minimum-Lateness Scheduling Recap

- There is an $O(n \log n)$ greedy algorithm for the minimum-lateness scheduling problem
  - Sort by earliest deadline and schedule jobs consecutively with no gaps
  - Analyze via an exchange argument



Network Design

- **Build a cheap, connected graph**

- We are given
  - A set of nodes $V = \{v_1, \dots, v_n\}$ and edges $E \subseteq V \times V$
  - a weight function on the edges w_e
- Want to build a network to connect the nodes
  - Every v_i, v_j must be connected
  - Must be as cheap as possible

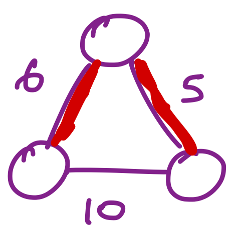
- **Many variants of network design**

Minimum Spanning Trees (MST)

- **Input:** a weighted graph $G = (V, E, \{w_e\})$
  - Undirected, connected, weights may be negative
  - All edge weights are distinct

- **Output:** a spanning tree T of minimum cost
  - A spanning tree of G is a subset of $T \subseteq E$ of the edges such that (V, T) forms a tree
  - Cost of a spanning tree T is the sum of the edge weights

input



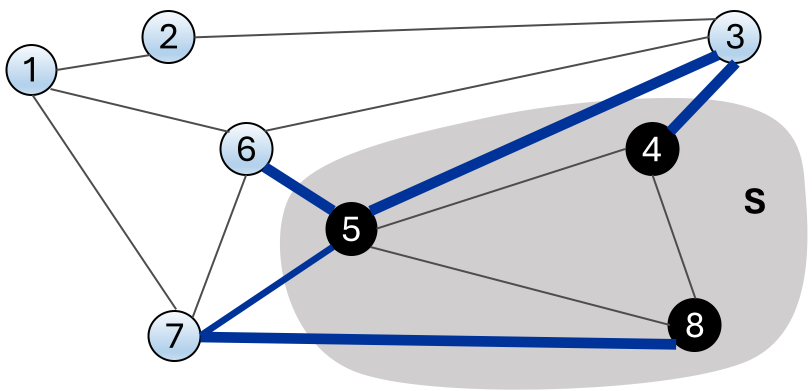
— tree edges

$\text{cost}(T) = \sum_{e \in T} w_e$

$\text{cost} = 11$

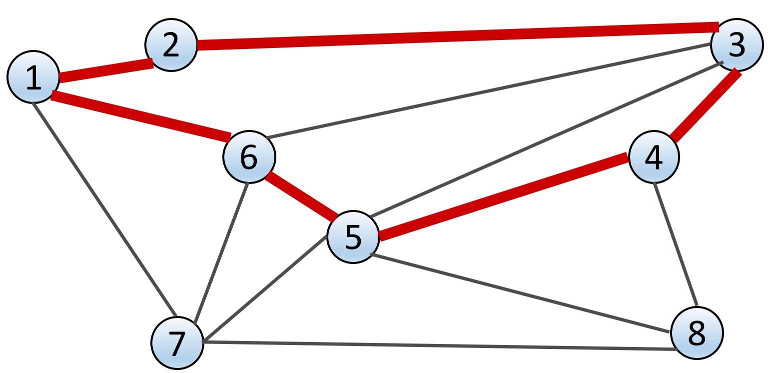
Cuts and Cycles

Cut: a subset of nodes S **Cutset:** edges w/ 1 endpoint in cut



Cut S = {4, 5, 8}
Cutset of S = (5,6), (5,7), (3,4), (3,5), (7,8)

Cycle: a set of edges $(v_1, v_2), (v_2, v_3), \ldots, (v_k, v_1)$



Cycle C = (1,2),(2,3),(3,4),(4,5),(5,6),(6,1)

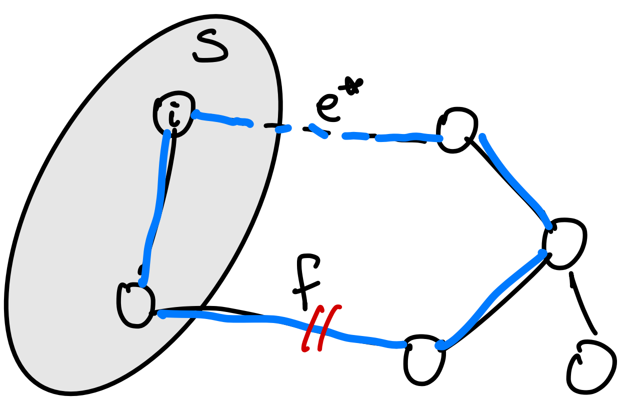
Cut Property

- If all edge weights are distinct, then there is a unique MST T^*

"safe edge"

Cut Property: For every cut S if e^* is the minimum weight edge in the cutset of S, then e^* is in T^*

Proof: Let T^* be the unique MST. Suppose $e^* \notin T^*$



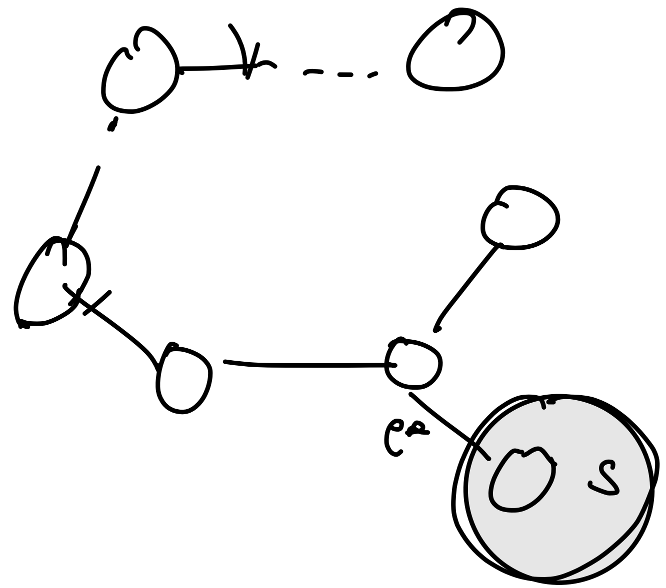
① Let $T' = T^* \cup \{e^*\}$

- T' has a cycle
- exists an edge $f \in T^*$ and in cutset(S) and $w_f > w_{e^*}$

② Let $T'' = T' \setminus \{f\}$

- T'' is a spanning tree
- $\text{cost}(T'') < \text{cost}(T^*)$

Minimum Spanning Trees (MST)



The "Only" MST Algorithm

$T = \emptyset$

While T is not a spanning tree:

Add one or more "safe" edges to T

Borůvka’s Algorithm

Borůvka’s Algorithm

Borůvka's Algorithm

Kruskal's Algorithm

Let $T = \emptyset$

Sort edges by weight $w_1 \leq w_2 \leq \dots w_m$

For each edge e in ascending order of wt:

  if $T + \{e\}$ has a cycle:

    continue

  else

    add e to T

How to implement efficiently?

Can implement so that the whole loop is $O(m \log m)$ time

MST Recap

- There is an $O(m \log n)$ greedy algorithm for finding a minimum spanning tree
  - There are actually several such algorithms
  - Bespoke analysis using structural properties of MSTs