Lecture 1: Greedy Algorithms

Yury Makarychev

Administrative

- Lectures: on Tuesdays and Thursdays, 9:30-10:50am
- Tutorials: on Wednesdays, 4:30-5:20pm

- TA: Theodoros Papamakarios
- Grader: Francisco Mendes

- Textbook: *Algorithm Design* by Kleinberg and Tardos
- Please make sure that you have access to the Canvas webpage for the course!

Assignments

4 homework assignments (60%) + 1 final exam (40%)

Each HW will have 3 problems + 1 programming assignment

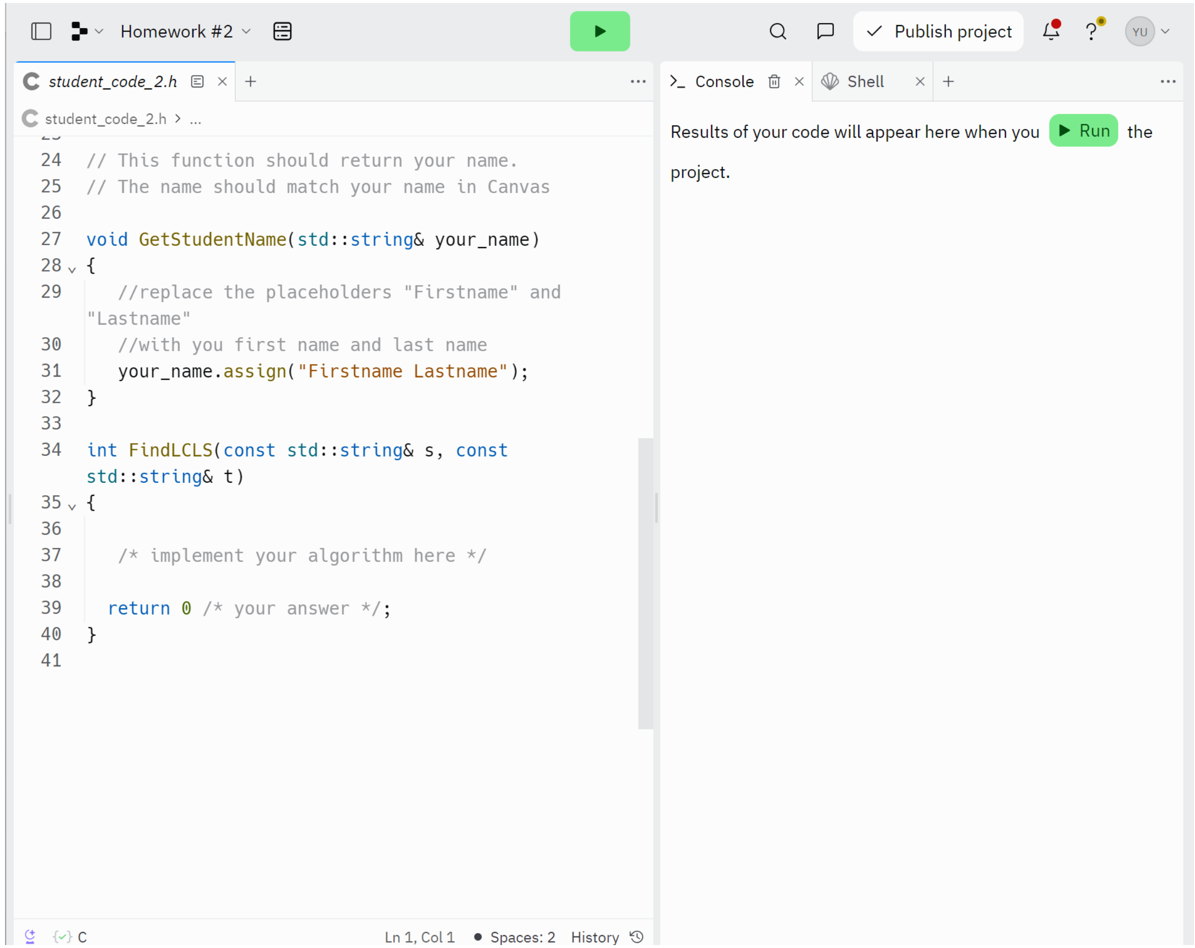
Please submit your text solutions via Gradescope and programming assignment solutions on Canvas.

- First HW will be posted next Thursday.

Assignments

Programming assignment:

- Need to design and implement an algorithm in C++.
- Use replit.com. You will need to implement only one function.



Questions?

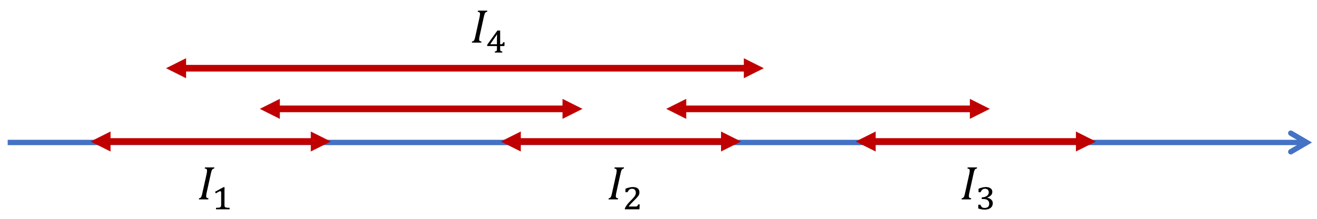
Syllabus

- Greedy Algorithms
- Dynamic Programming
- Max Flow, Min Cut, and Their Applications
- Linear Programming
- Classes P and NP. NP-hardness.
- Approximation Algorithms
- Multiplicative Weight Updates
- (if we have any time left) more advanced topics

Greedy Algorithms

Job (Interval) Scheduling Problem

➢ Given a set of intervals $I_1 = (s_1, t_1), \dots, I_n = (s_n, t_n)$ on the real line.



➢ Interval I_i represents a job that starts at s_i and finishes at t_i

➢ Need to schedule a subset of jobs on a machine/computer that can execute only one job at a time

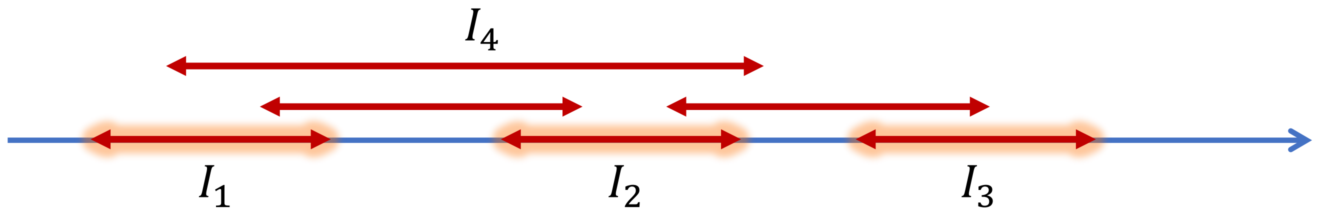
➢ Two jobs I_i and I_j are *compatible* if they don't overlap:

$$\text{either} \quad t_i \leq s_j \quad \text{or} \quad t_j \leq s_i$$

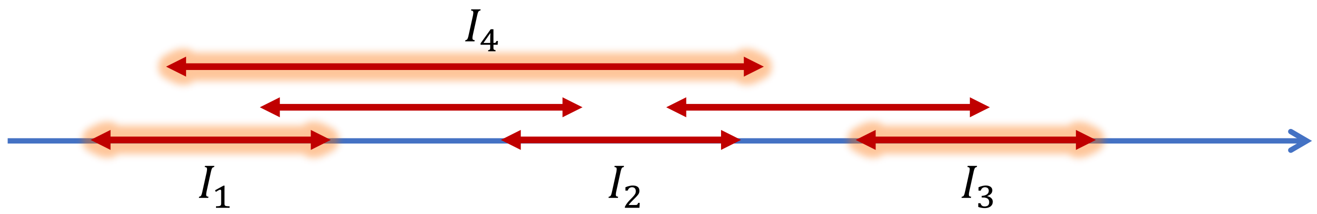
Job Scheduling Problem: Examples

A *feasible* solution/schedule is a subset of jobs $S \subseteq \{I_1, \ldots, I_n\}$ such that every two jobs in S are compatible.

$\{I_1, I_2, I_3\}$ is a feasible solution:



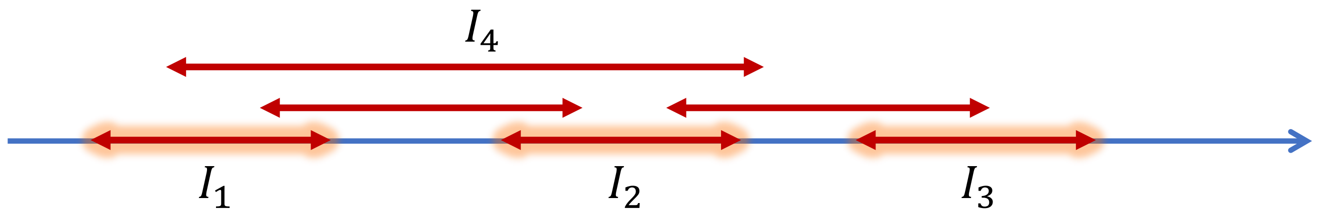
$\{I_1, I_3, I_4\}$ is not a feasible solution, because I_1 and I_4 overlap:



Job Scheduling Problem

- Given a set of jobs $I_1 = (s_1, t_1), \dots, I_n = (s_n, t_n)$.
- Find a feasible solution S of maximal size.

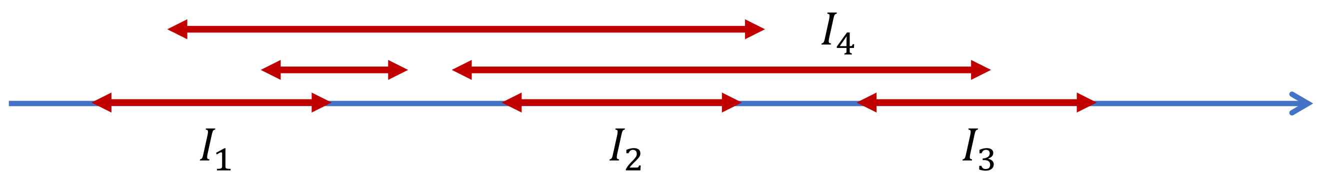
I.e., find the largest number of intervals no two of which overlap.



Greedy Algorithm

We will design a *greedy algorithm*.

A greedy algorithm constructs an optimal solution step-by-step. At each step, it makes a choice that is “locally optimal”.

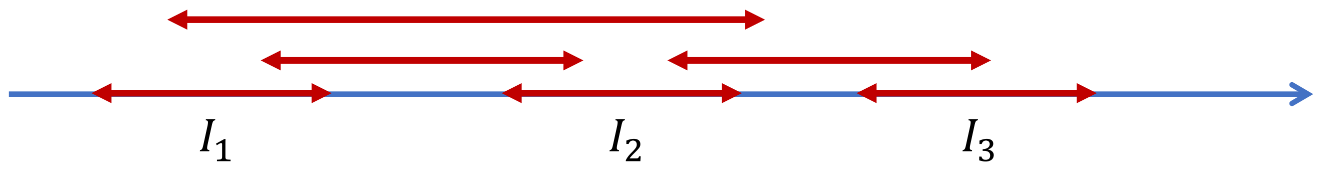


Greedy Algorithm

We will design a *greedy algorithm*.

A greedy algorithm constructs an optimal solution step-by-step. At each step, it makes a choice that is “locally optimal”.

- $S = \emptyset$
- while there is a job that starts after all jobs in S finish
  - among all such jobs, find a job I_j with the least value of t_j (that finishes first)
  - add I_j to S



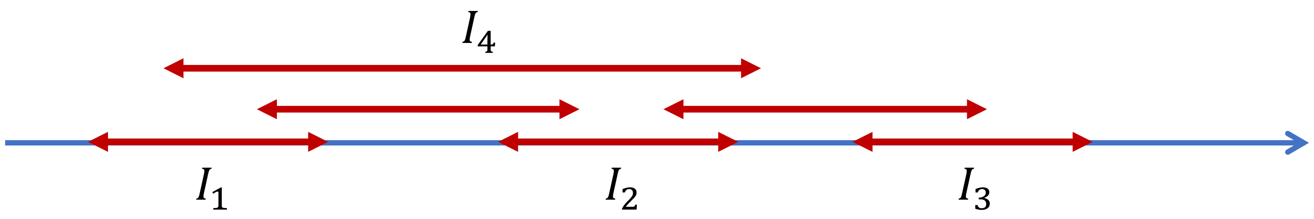
TO DO items

- Prove that the algorithm finds a feasible solution.
- Prove that the algorithm finds an optimal solution:

$$|S| \geq |S'| \text{ for every feasible solution } S'$$

- Discuss how to implement the algorithm efficiently and find its running time.

Feasibility



- $S = \emptyset$
- while there is a job that starts after all jobs in S finish
  - among all such jobs, find a job I_j with the least value of t_j (that finishes first)
  - add I_j to S

Why is the set of jobs S returned by the algorithm a feasible solution?

Proof: Consider two jobs I_i and I_j in S.

Assume that I_i was added to S before I_j.

When we added I_j, it was compatible with all jobs in S, including I_i. ■

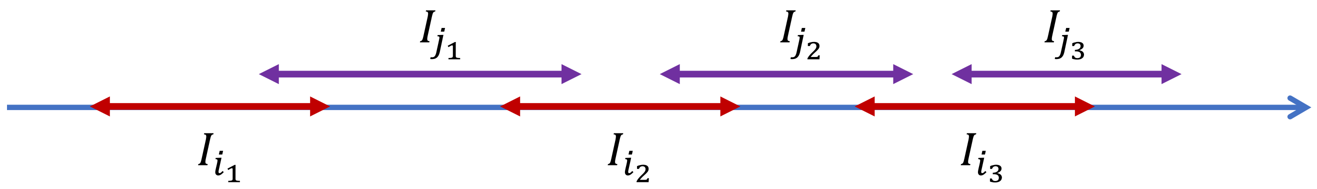
Optimality

Consider an optimal solution S^*. Prove that $|S| = |S^*|$.

Proof:

Sort jobs in S and S^* from left to right. Let

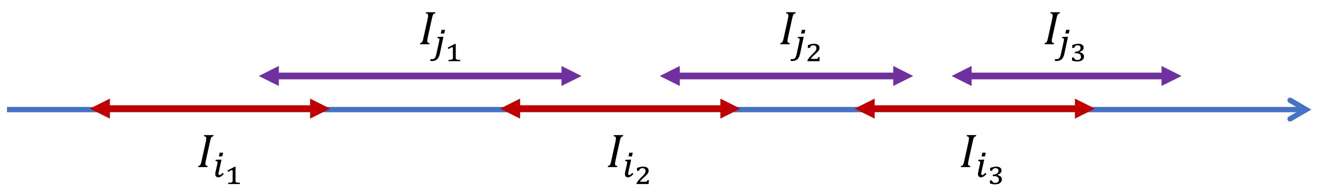
$$S = \{I_{i_1}, \dots, I_{i_k}\} \text{ and } S^* = \{I_{j_1}, \dots, I_{j_{k^*}}\}$$



Note $k \le k^*$, since S^* is an optimal solution. Want: $k \ge k^*$.

Optimality

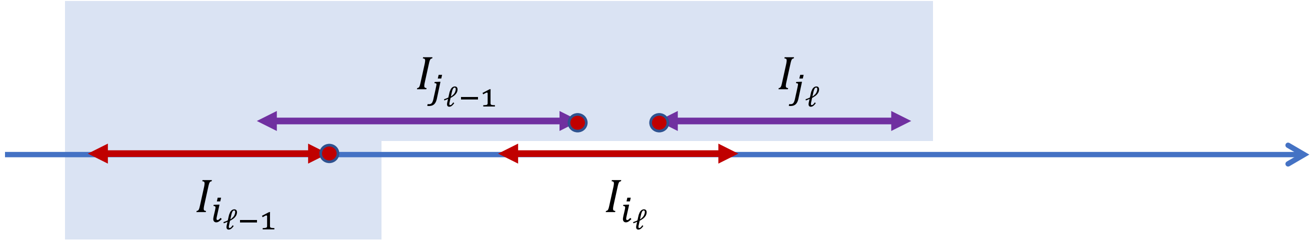
Prove by induction on ℓ that $t_{i_\ell} \leq t_{j_\ell}$ for $\ell \in \{1, \ldots, k\}$.



Base case: $t_{i_1} \leq t_{j_1}$. Why?

Optimality

Prove by induction on ℓ that $t_{i_\ell} \leq t_{j_\ell}$ for $\ell \in \{1, \ldots, k\}$.



Induction step: Assume that $t_{i_{\ell-1}} \leq t_{j_{\ell-1}}$.

- I_{j_ℓ} lies to the right of $I_{i_{\ell-1}}$: $s_{j_\ell} \geq t_{j_{\ell-1}} \geq t_{i_{\ell-1}}$.
- When we added I_{i_ℓ} to S, I_{j_ℓ} was to the right of $I_{i_{\ell-1}}$. We could have added I_{j_ℓ} but instead added I_{i_ℓ}. Why?

Optimality

Prove by induction on ℓ that $t_{i_\ell} \leq t_{j_\ell}$ for $\ell \in \{1, \dots, k\}$.

Induction step: Assume that $t_{i_{\ell-1}} \leq t_{j_{\ell-1}}$.

- I_{j_ℓ} lies to the right of $I_{i_{\ell-1}}$: $s_{j_\ell} \geq t_{j_{\ell-1}} \geq t_{i_{\ell-1}}$.
- When we added I_{i_ℓ} to S, I_{j_ℓ} was to the right of $I_{i_{\ell-1}}$. We could have added I_{j_ℓ} but instead added I_{i_ℓ}. Why?
- It had to be the case that: $t_{i_\ell} \leq t_{j_\ell}$.
- We proved $t_{i_\ell} \leq t_{j_\ell}$ for $\ell \in \{1, \dots, k\}$. ■

Optimality

Proved by induction that $t_{i_\ell} \leq t_{j_\ell}$ for $\ell \in \{1, \ldots, k\}$.

Remains to show that $k \geq k^*$. Assume to the contrary that $k < k^*$.

- $S = \emptyset$
- while there is a job that starts after all jobs in S finish
  - …

The algorithm terminates after adding I_{i_k} to S.

Thus, there is no job to the right of I_{i_k} in S. However, $s_{j_{k+1}} \geq t_{j_k} \geq t_{i_k}$.

Contradiction. We proved that S is an optimal solution.

Implementation

- Sort all jobs by their stop time t_i:
$$t_1 \leq t_2 \leq \cdots \leq t_n$$
- Add I_1 to S
- $T = t_1$ *(the termination time of the last currently scheduled job)*
- $i = 2$ *(the first unprocessed job)*
- while $i \leq n$ do
    - if $s_i \geq T$, then
        - add I_i to our schedule S
        - $T = t_i$
    - $i = i + 1$

Implementation

Running time

- Sort all jobs by their stop time t_i:
$$t_1 \leq t_2 \leq \cdots \leq t_n$$
- Add I_1 to S
- $T = t_1$ *(the termination time of the last currently scheduled job)*
- $i = 2$ *(the first unprocessed job)*
- while $i \leq n$ do
  - if $s_i \geq T$, then
    - add I_i to our schedule S
    - $T = t_i$
  - $i = i + 1$

$O(n \log n)$

$O(n)$

Combinatorial Optimization

Job Scheduling is a combinatorial optimization problem.

In a combinatorial optimization problem:

- We are given an instance of the problem, which defines the set of feasible solutions and an objective function.
- The goal is to find an optimal solution, a feasible solution whose objective is
  - greater than or equal to (for a maximization problem), or
  - smaller than or equal to (for a minimization problem)

than the value of any other feasible solution.

↑ value

↓ cost

Combinatorial Optimization

Job Scheduling is an example of a large class of scheduling problem. It's also known as Maximum Independent Set in Interval Graphs.

In this class, we will mostly study various approaches – such as Greedy Algorithms, Dynamic Programming, and Linear Programming – for solving combinatorial optimization problems.

Today, we designed a greedy algorithm for Job Scheduling. We used a "stay ahead" type argument: our solution always "stays ahead" of an optimal solution ($t_{i_\ell} \leq t_{j_\ell}$).

Greedy Algorithms: Pros and Cons

+ Very efficient
+ Easy to implement and analyze
+ If there is a greedy algorithm for a problem, use it!

- There are no greedy algorithms for many problems
- Even if there is a greedy algorithm for a problem, there may be no algorithm for its variant (e.g. for Weighted Job Scheduling).

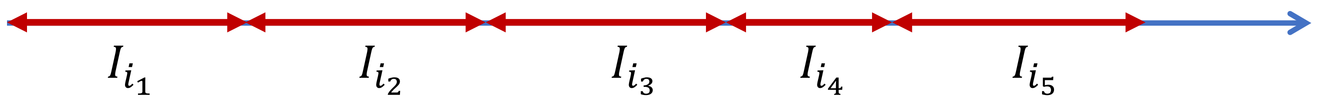
Minimum Weighted Completion Time

- Given n jobs.

 Each job i has duration t_i and weight w_i.

Jobs don't have fixed start and stop times.

- We want to run all n jobs one after another:



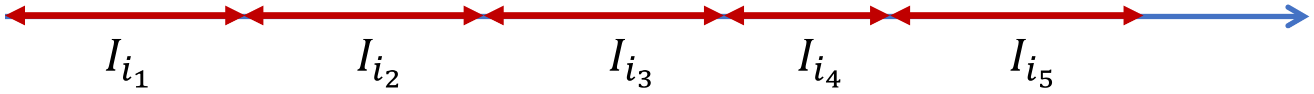
It's up to us to decide the order in which we run the jobs.

Minimum Weighted Completion Time

Given an order $i_1, \dots, i_n$, we schedule the jobs as follows:

- job i_1 starts at time 0 and completes at t_{i_1}
- job i_2 starts at time t_{i_1} and completes at $t_{i_1} + t_{i_2}$
- …

completion times



Completion time of job i_ℓ is

$$c(i_\ell) = \sum_{a=1}^{\ell} t_{i_a}$$

$$c(i_\ell) = \sum_{a=1}^{\ell} t_{i_a}$$

Objective

Note that $c(i)$ depends on the ordering $i_1, \dots, i_\ell$.

➢ Find an ordering $i_1, \dots, i_n$ that minimizes the weighted sum of completion times:

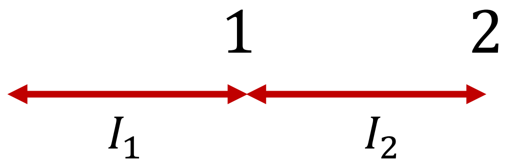
$$\sum_{i=1}^{n} w_i c(i)$$

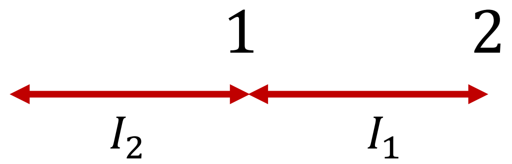
$$c(i_\ell) = \sum_{a=1}^{\ell} t_{i_a}$$

Warm Up

Consider a couple of examples. Let $n = 2$.

1. Assume that $t_1 = t_2 = 1$. There are two solutions:





Their costs are

$w_1 + 2w_2$ and $w_2 + 2w_1$

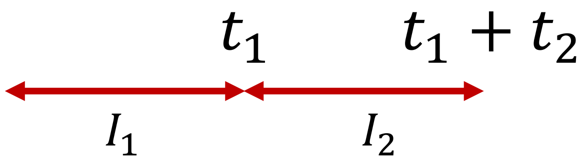
It’s better to put **heavier** jobs first.

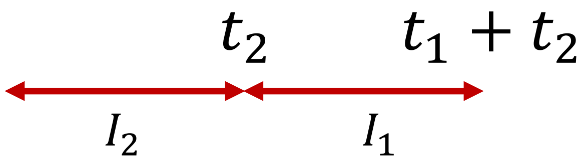
Warm Up

$$c(i_\ell) = \sum_{a=1}^{\ell} t_{i_a}$$

Consider a couple of examples. Let $n = 2$.

2. Assume that $w_1 = w_2 = 1$. There are two solutions:





Their costs are

$2t_1 + t_2$ and $t_1 + 2t_2$

It's better to put shorter jobs first.

Algorithm

Guess: sort all jobs according to t_i/w_i:

$$\frac{t_{i_1}}{w_{i_1}} \le \frac{t_{i_2}}{w_{i_2}} \le \cdots \le \frac{t_{i_n}}{w_{i_n}}$$

We prove that $i_1, \dots, i_n$ is an optimal solution.

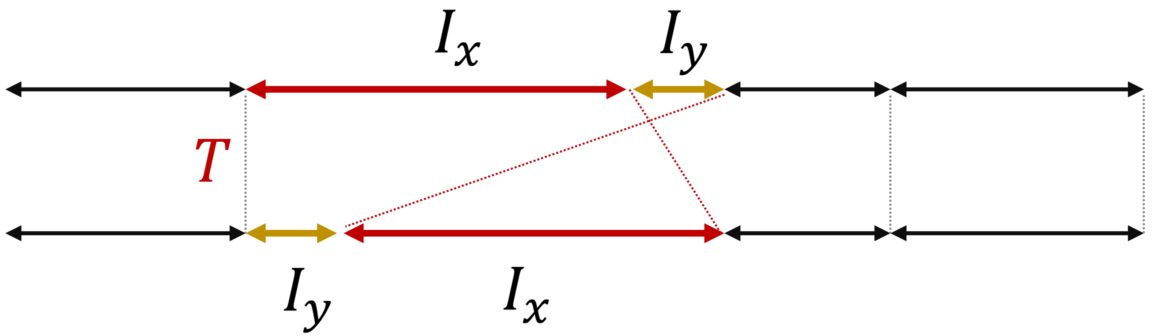
Consider two solutions:

$$j_1, \dots, \boxed{j_a, j_{a+1}}, \dots, j_n \text{ and } j_1, \dots, \boxed{j_{a+1}, j_a}, \dots, j_n$$

that differ only by the ordering of two jobs, j_a and j_{a+1}.

Algorithm

Denote $x = j_a$ and $y = j_{a+1}$



$c(b) = c'(b)$ for all $b \notin \{x, y\}$

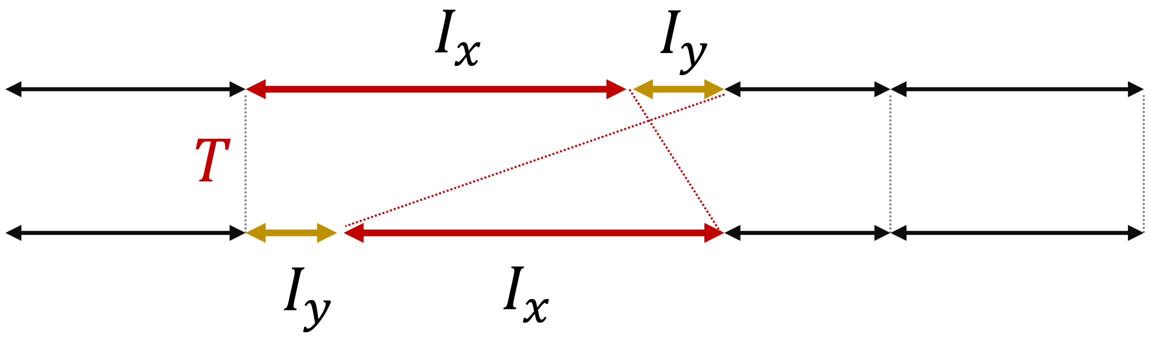
$c(x) = T + t_x$ $\quad$ $c'(x) = T + t_x + t_y$ $\quad$ $c(x) - c'(x) = -t_y$

$c(y) = T + t_x + t_y$ $\quad$ $c'(y) = T + t_y$ $\quad$ $c(y) - c'(y) = t_x$

$$\text{cost} - \text{cost}' = \sum_b w_b c(b) - \sum_b w_b c'(b) =$$

$$\sum_b w_b \big(c(b) - c'(b)\big) = -w_x t_y + w_y t_x$$

Algorithm



$\text{cost} - \text{cost}' = -w_x t_y + w_y t_x \leq 0$ if and only if $\frac{t_x}{w_x} \leq \frac{t_y}{w_y}$.

The first solution is better if and only $\frac{t_x}{w_x} < \frac{t_y}{w_y}$.

Algorithm

Consider an optimal solution $i_1^*, \ldots, i_n^*$.

Is it possible that $\frac{t_{i_a^*}}{w_{i_a^*}} > \frac{t_{i_{a+1}^*}}{w_{i_{a+1}^*}}$ for some a?

No! Then $i_1^*, \ldots, i_n^*$ would not be an optimal solution!

Algorithm

Consider an optimal solution $i_1^*, \ldots, i_n^*$.

We conclude that

$$\frac{t_{i_1^*}}{w_{i_1^*}} \leq \frac{t_{i_2^*}}{w_{i_2^*}} \leq \cdots \leq \frac{t_{i_n^*}}{w_{i_n^*}}$$

as required.

Are we done?

Running time?

Summary

- Discussed greedy algorithms
- Designed algorithms for Interval/Job Scheduling and Minimum Weighted Completion Time