LEC013 Greedy: Scheduling, MST

VG441 SS2021

Cong Shi
Industrial & Operations Engineering
University of Michigan

Scheduling Problem

- We are given n jobs to schedule
- For each job $j = 1, \ldots, n$, let
  - w_j be the weight (or the importance)
  - l_j be the length (or the time required)

- Define the completion time of job j

  $c_j =$ sum of the lengths of jobs up to and including l_j

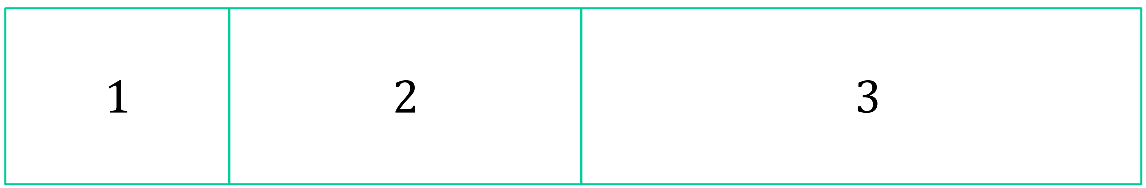
Objective: to minimize the weighted sum of completion times

$$\sum_{j=1}^{n} w_j \cdot c_j$$

Intuition

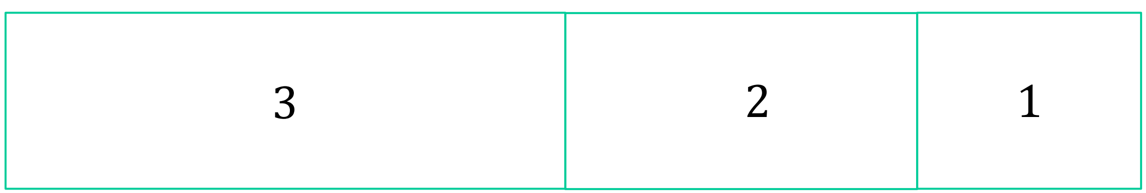
- If all jobs have the same length, we prefer larger weighted jobs to appear earlier in the order
- If all jobs have equal weights, we prefer shorter length jobs to appear earlier in the order

$w_i = 1 \ \forall i$



$$w_1c_1 + w_2c_2 + w_3c_3 = 1 + 3 + 6 = 10$$

$w_i = 1 \ \forall i$

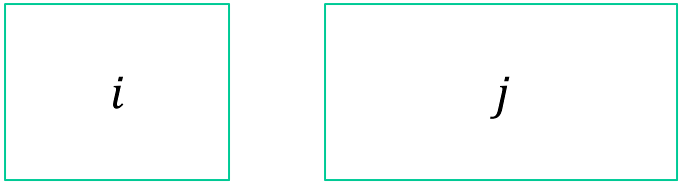


$$w_1c_1 + w_2c_2 + w_3c_3 = 3 + 5 + 6 = 14$$

Quandrum (Tricky Cases)

- What do we do in the cases where

$$l_i < l_j \text{ and } w_i < w_j$$



- Idea: give a priority score (prefers smaller length and larger weight at the same time)

Guess #1

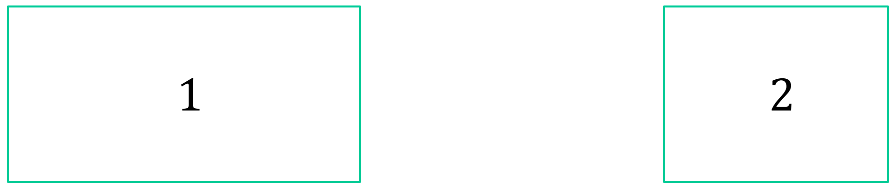
$$\text{score-diff} = l_j - w_j$$

$$\text{score-ratio} = l_j / w_j$$

Guess #2

Quandrum (Tricky Cases)

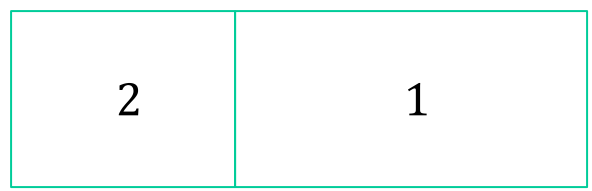
- Let a look at a simple example: $l_i < l_j$ and $w_i < w_j$



$l_1 = 5$ and $w_1 = 3$ $\qquad$ $l_1 = 2$ and $w_1 = 1$

- Score-diff does not work so well…

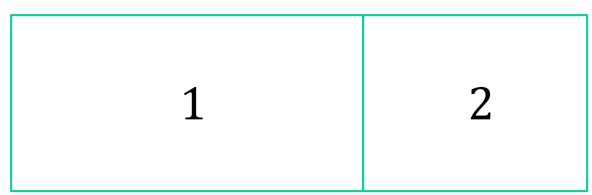
$\text{score-diff} = l_j - w_j$



$\text{WC} = 2 + 21 = 23$

- Score-ratio seems to work

$\text{score-ratio} = l_j / w_j$



$\text{WC} = 15 + 7 = 22$

Correctness Argument

- Claim: Ranking by score_ratio is correct.
- Proof (by exchange argument):

Consider some input of n jobs, now rename the jobs according the score_ratio, then our greedy algo picks the schedule

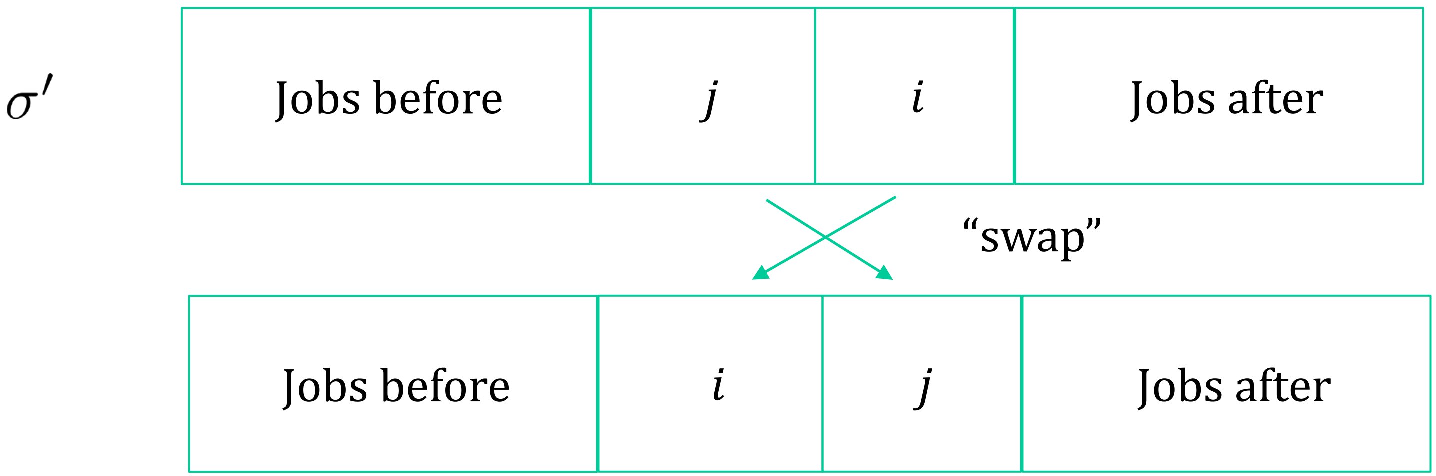
$$\sigma = 1, 2, 3 \ldots, n \text{ with } \frac{l_1}{w_1} \leq \frac{l_2}{w_2} \leq \frac{l_3}{w_3} \leq \ldots \leq \frac{l_n}{w_n}$$

Correctness Argument

- Consider any other schedule σ', we want to show that σ is as good as σ'

Now if $\sigma' \neq \sigma$ then at some point in σ' , there exists a job i *right after* a job j with $i < j$ (why?)

Correctness Argument



$$WC^{\text{after swap}} - WC^{\text{before swap}} = w_j l_i - w_i l_j \leq 0$$

After at most (n-choose-2) steps, we recover our greedy σ with at least good as σ'

σ

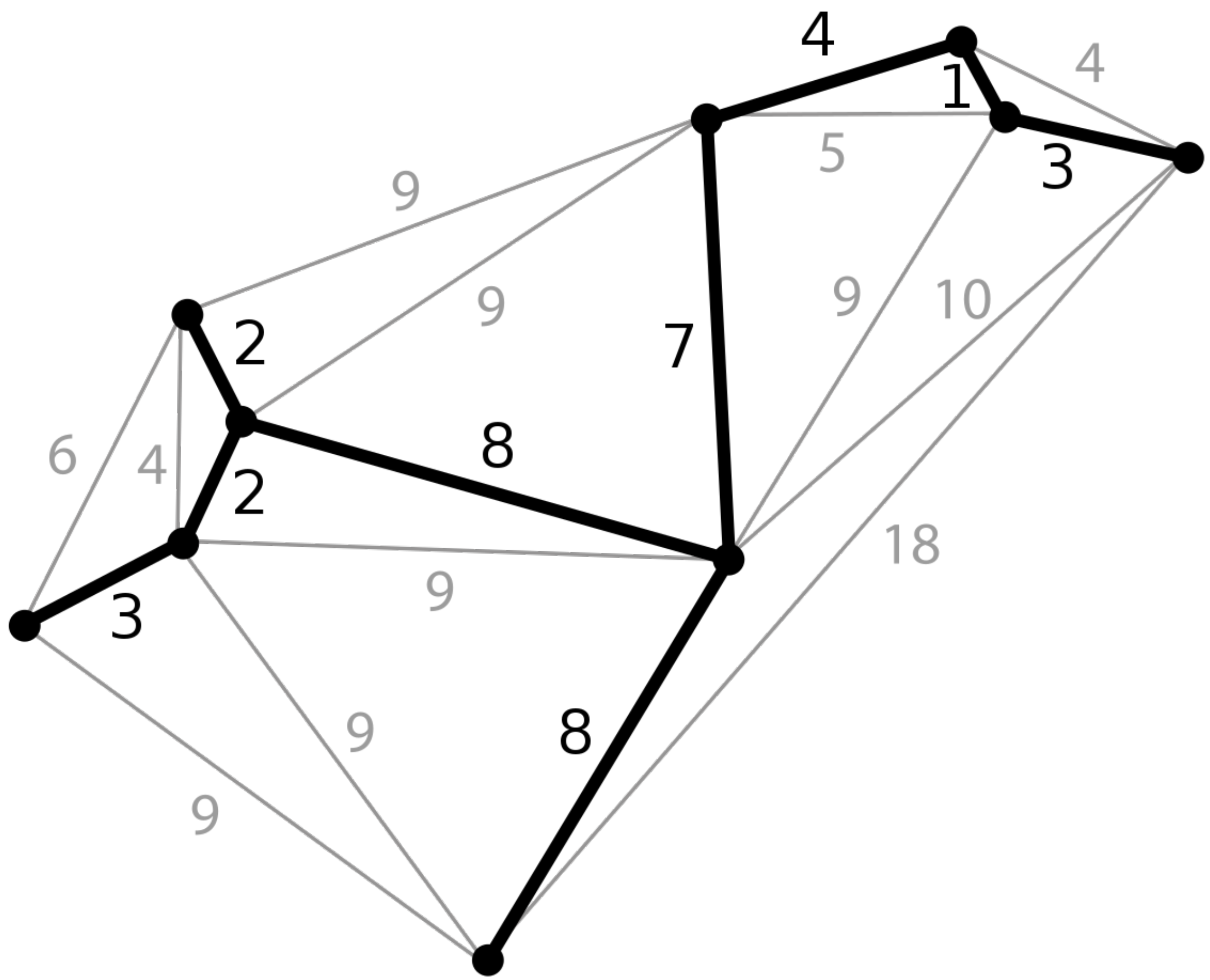
MST

Given undirected connected graph, $G = (V, E)$. There is a function $c : E \to \mathbb{R}$, showing the cost of each edge. Assume that there are m edges in G

Definition (tree): $T = (V_T, E_T)$, is an undirected graph in which any two vertices are connected by exactly one path. In other words, any acyclic connected graph is a tree.

Definition (a spanning tree): $T = (V_T, E_T)$ in graph $G = (V, E)$ is a tree with $E_T \subseteq E$ that has all vertices covered, i.e. $V_T = V$

An Example of MST



Greedy Algorithm for MST (Kruskal Algo)

1. sort edges in non-decreasing order of cost, $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$
2. Let $T_1 = \emptyset$
3. For $i = 1, 2, \cdots, m$, if $(T_i \cup \{e_i\})$ contains no cycle, let $T_{i+1} = T_i \cup \{e_i\}$

We apply quicksort to assign indices to edges in the first step. We can apply a simple check of whether there is a cycle in graph $T_i \cup e_i$ in step 3

1. Get both ends of edge e_i, vertex a and b
2. Make a list L that contains all vertices connected to a in T_i
3. If vertex b is in L, there will be a cycle in $T_i \cup \{e_i\}$. If b is not in L, then $T_i \cup \{e_i\}$ is acyclic.

Note that the algorithm above takes $O(m)$ time. So the overall greedy algorithm runs in polynomial time.

Python Code

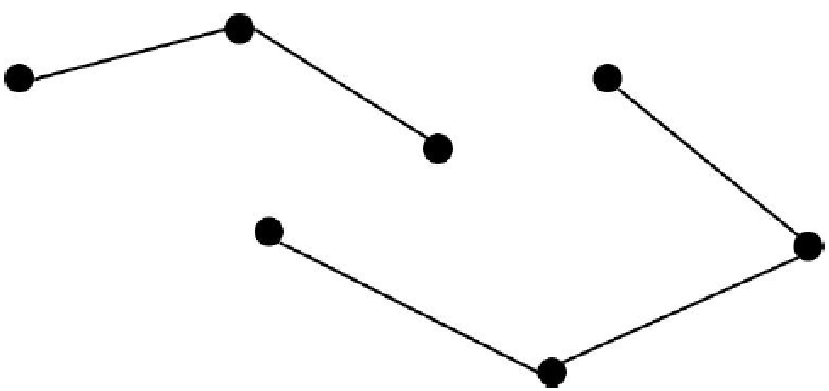
```
input graph               minimum spanning tree

     (0)                         (0)
    /   \                       /
   3     8                     3
  /       \                   /
(3)---5---(1)               (3)---5---(1)
  \       /                           /
   6     2                           2
    \   /                           /
     (2)                         (2)
```

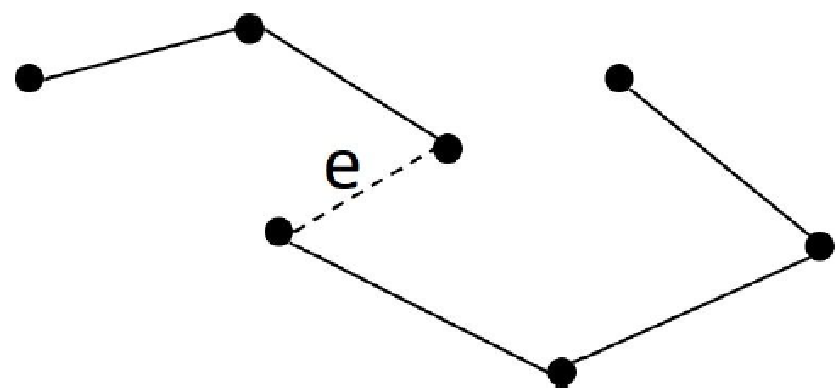
```
>>> from scipy.sparse import csr_matrix
>>> from scipy.sparse.csgraph import minimum_spanning_tree
>>> X = csr_matrix([[0, 8, 0, 3],
...                 [0, 0, 2, 5],
...                 [0, 0, 0, 6],
...                 [0, 0, 0, 0]])
>>> Tcsr = minimum_spanning_tree(X)
>>> Tcsr.toarray().astype(int)
array([[0, 0, 0, 3],
       [0, 0, 2, 5],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

Lemma: The algorithm gives a tree T that covers all vertices in G.

Proof: Given the graph is connected, we can prove this property by contradiction. If in the tree T generated by algorithm, not all the vertices are connected, there exist an edge $e \in E$ between two disconnected components of T (see Figure 1 a). Because T is acyclic, and two components are not connected, we can declare that $T \cup \{e\}$ is also acyclic. Then according to the algorithm, e must have been included.



(a) T



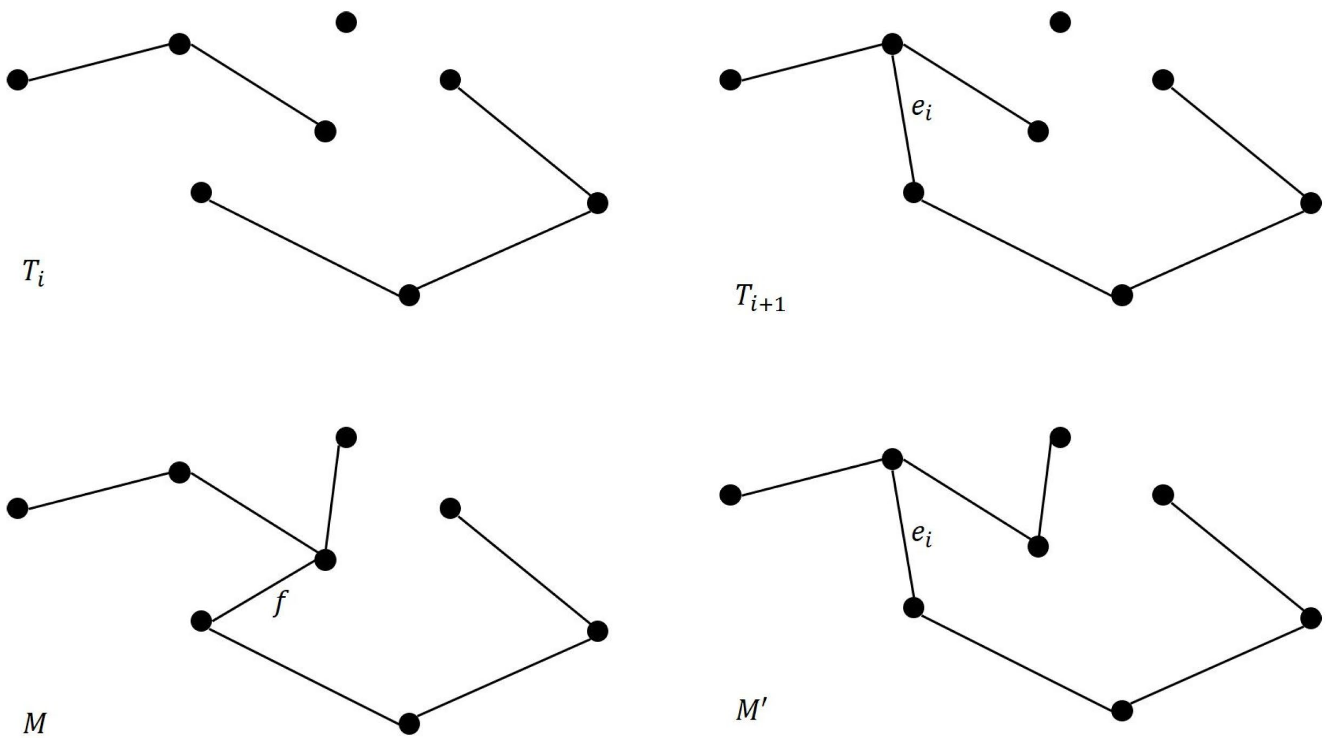
(b) $T \cup \{e\}$

Analysis

Let $i = 1, 2, \cdots, m, m+1$ denote the iteration in the algorithm. Let T_i denote the solution at the beginning of i^{th} iteration.

Lemma: For each iteration i, there is a minimum spanning tree M with $T_i \subset M$

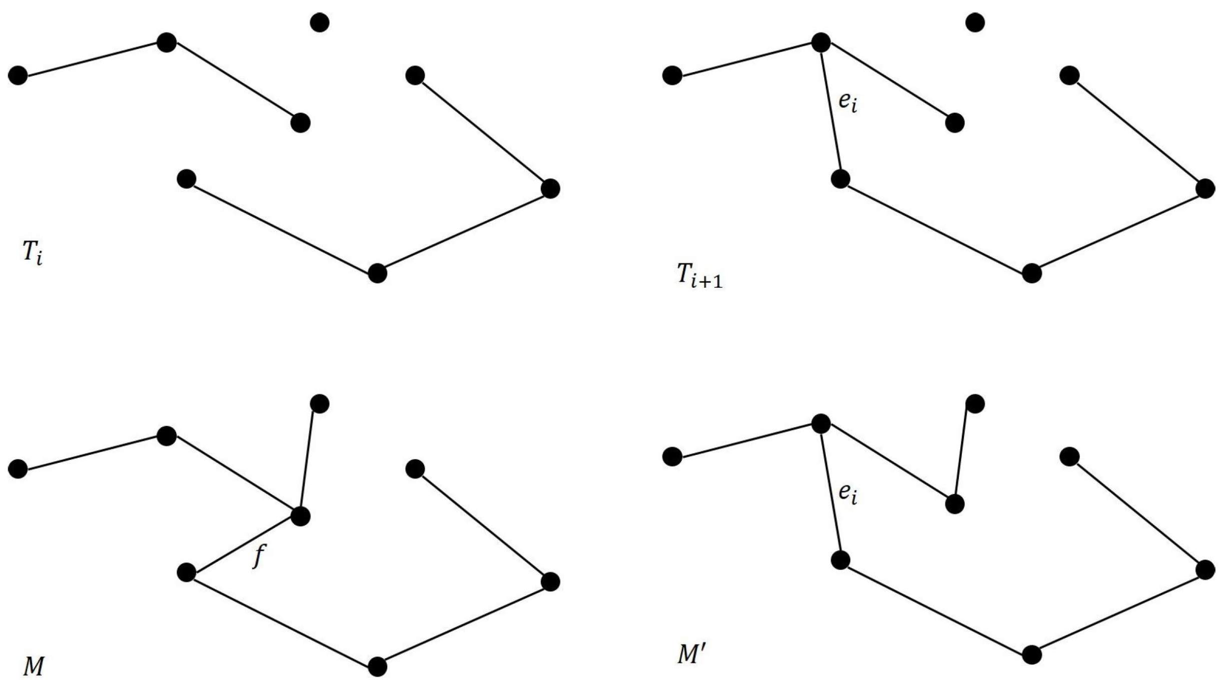
Quandrum (tricky case) Suppose $T_i \subseteq M$ but $T_{i+1} = T_i \cup \{e_i\} \nsubseteq M$



$$\text{cost}\,(M') = \text{cost}\,(M \backslash \{f\} \cup \{e_i\}) = \text{cost}(M) - \text{cost}(f) + \text{cost}\,(e_i) \leq \text{cost}(M)$$

Analysis

Suppose $T_i \subseteq M$ but $T_{i+1} = T_i \cup \{e_i\} \nsubseteq M$



$$\text{cost}\,(M') = \text{cost}\,(M \backslash \{f\} \cup \{e_i\}) = \text{cost}(M) - \text{cost}(f) + \text{cost}\,(e_i) \leq \text{cost}(M)$$

- $M \cup e_i$ must contain a cycle (if not, then M is disconnected.)
- There must be an edge f in this cycle that is costlier than e_i (if not, then the greedy algorithm would not pick e_i)