Approximation algorithms

If not able to compute an optimal solution, go for an approximately optimal solution.

$2, 1.5, 1+\varepsilon$

Minimization problem : your solution cost $\leq \alpha \cdot$ optimal cost

Maximization problem : your solution value $\geq \beta \cdot$ optimal value

$0.5, 0.9, 1-\varepsilon$

Load balancing

m processors (identical)

n jobs with processing times $t_1, t_2, t_3, \dots, t_n$

Assign jobs to processors. (not splitable)

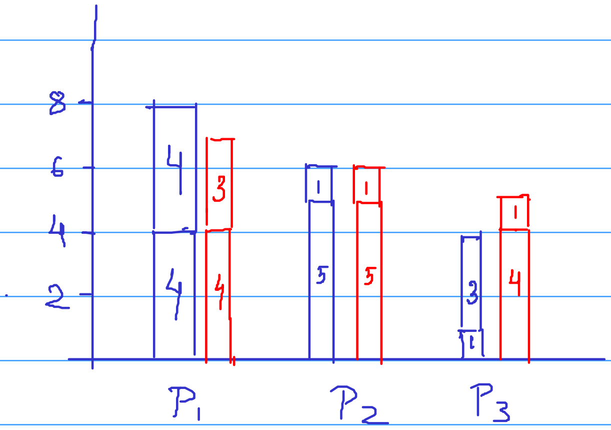
Want to finish all the jobs as soon as possible.

Minimize makespan

maximum total load of any processor.

Example $m = 3$ processors

jobs $\rightarrow 4, 4, 5, 1, 1, 3$



8, 6 4

makespan

7 6 5

Greedy algorithm

For $i = 1$ to n

assign job i to the processor which has the minimum load currently.

Does greedy always give an optimal solution ?

4, 4, 5, 1, 1, 3

P_1	P_2	P_3
4	4	5
1	1	3
5	5	8

doesn't give optimal

Is there a better order for the greedy algorithm.

sort in decreasing order

5, 4, 4, 3, 1, 1

P_1	P_2	P_3
5	4	4
	3	1
		1
5	7	6

HW find an example where this algorithm is not optimal

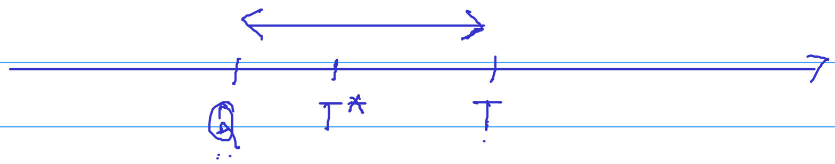
→ There is no polynomial time algorithm known for minimizing makespan.

Suppose T^* is the optimal makespan.

Claim: Greedy algorithm (arbitrary order) gives a solution with makespan $T \leq 2T^*$ [Graham 1966]

2- approximation algorithm.

How can we prove such a bound when we don't have any idea about T^*



Find some natural lower bounds for T^* and relate them with T.

Lower bounds on T^*

① $T^* \geq$ average load of a processor (Q_1)

$$= \sum_{i=1}^{n} t_i / m$$

② $T^* \geq \max\{t_1, t_2, \cdots t_n\}$ (Q_2)

~~Greedy $\leq 2 \cdot \max\{t_1, \ldots, t_n\} \leq 2T^*$~~

counter example 2 processors, Jobs - 6, 7, 5, 5, 6, 8

~~Greedy $\leq 2 \cdot$ average load of a processor~~

counter example 3 processors, Jobs - 4, 4, 4, 30

Claim : Load on any processor by Greedy algorithm

$\leq$ Max processing time of any job + Average load of a processor

$(Q_2 + Q_1)$

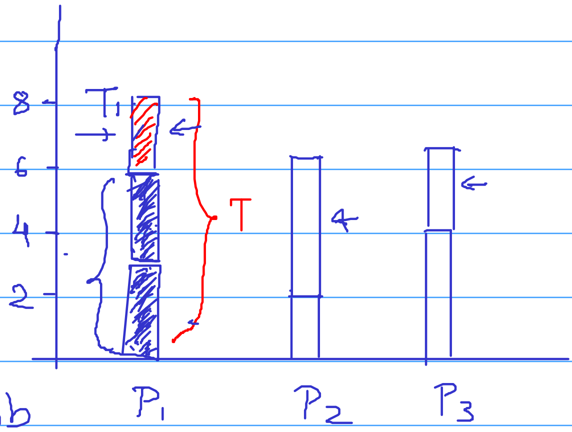
Claim implies

$$T \leq T^* + T^* = 2T^*$$

$\Rightarrow$ 2- approximation.

Oct 21

Proof the Claim:



T = the last load assigned (T_1)
+
Total load before the last load (T_2)

$T_1 \leq$ max processing time of a job

T_2 = minimum total load of any processor at that time

$\leq$ avg total load among all processors at that time

$\leq Q_1$

When we assign a job to a processor it current total load is minimum among all processors.

$$\Rightarrow \quad T \leq Q_1 + Q_2 \leq 2T^*$$

Que: Is our analysis tight?

It is possible that the greedy algorithm always gives, say 1.5 approximation, but our analysis is weak.

Try to construct an example where greedy (arbitrary order) gives a solution with makespan $\simeq 2T^*$

HW

2 processors 4, 4, 4, 30

$T^* = 30$ $T = 34$

What about the greedy algorithm with decreasing order of processing times.

Sort the jobs in decreasing order of processing times
For $i = 1$ to n
    assign job i to the processor which has the minimum load currently.

Claim: Greedy algorithm with decreasing order of processing times is a 3/2 - approximation algorithm

$$T \leq 3/2\, T^*$$

Where can we improve the previous argument.

Can we show

T_1 = last load assigned $\leq \frac{1}{2} T^*$

No, if only one job assigned.

Yes if there are at least two jobs assigned on the processor

Final analysis: Two cases

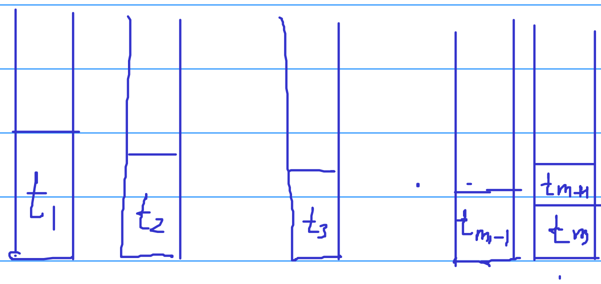
① only one job assigned to the processor

$$T \leq \text{max processing time} \leq T^*$$

(2) At least two jobs assigned to the processor

Then there must be at least m+1 jobs in total
because greedy will assign first m jobs to m different processors

jobs processing times
→ $t_1 \geq t_2 \geq t_3 \cdots \geq t_m \geq t_{m+1} \geq \cdots$



Last job assigned $\leq t_{m+1}$

$T^* \geq 2\, t_{m+1} \implies$ last job assigned $\leq t_{m+1} \leq \frac{T^*}{2}$

because in the optimal solution, there must be some processor that takes at least two jobs from first m+1

T = last job assigned + total load before the last job

$$\leq \frac{T^*}{2} + T^*$$

$$\leq \frac{3T^*}{2}$$

Further Improvements:

→ Greedy with decreasing order can be actually shown to be 4/3 approximation.

→ More sophisticated techniques give arbitrarily small approximation factor

$1+\varepsilon$ for any $\varepsilon > 0$
but running time $O(n^{1/\varepsilon^2})$