Approximation algorithms

If not able to compute an optimal solution, go for an approximately optimal solution. $^2, 1.5, 1+\epsilon$

Minimization problem: your solution cost $\leq \alpha \cdot$ optimal cost
Maximization problem: your solution value $\geq \beta \cdot$ optimal value $^2, 0.5, 0.9, 1-\epsilon$

Load balancing

$m$ processors (identical)

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

Assign jobs to processors (not splittable)
Want to finish all the jobs as soon as possible.

Minimize makespan

\[ \text{maximum total load of any processor} \]

Example $m = 3$ processors

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

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

\[ P_1 \quad P_2 \quad P_3 \]
\[
4 \quad 4 \quad 5 \\
1 \quad 1 \quad 3 \\
5 \quad 5 \quad 8
\]
doesn't give optimal

Is there a better order for the greedy algorithm?

sort in decreasing order

\[ P_1 \quad P_2 \quad P_3 \]
\[
5 \quad 4 \quad 4 \\
3 \quad 1 \quad 1
\]

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. \[ T^* \geq \text{average load of a processor } (Q_1) = \frac{\sum_{i=1}^{n} t_i}{m} \]

2. \[ T^* \geq \max \{ t_1, t_2, \ldots, t_n \} \]  \hspace{1cm} (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** \[ \leq 2 \cdot \text{average load of a processor} \]

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

**Claim**: Load on any processor by Greedy algorithm

\[ \leq \text{Max processing time of any job} + \text{Average load of a processor } (Q_2 + Q_1) \]

Claim implies

\[ T \leq T^* + T^* = 2T^* \]

\[ \Rightarrow \] 2-approximation.
Proof the Claim:

\[ T = \text{the last load assigned (} T_i \text{)} + \text{Total load before the last load (} T_2 \text{)} \]

\[ T_1 \leq \text{max processing time of a job } P_i \quad P_1 \quad P_2 \quad P_3 \]

\[ T_2 = \text{minimum total load of any processor at that time} \]

\[ \leq \text{any total load among all processors at that time} \]

\[ \leq Q_1 \]

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

\[ \Rightarrow T \leq Q_1 + Q_2 \leq 2T^* \]

---

**Q:** 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 \( \approx 2T^* \)

**HW**

2 processors 4, 4, 4, 30

\[ T^* = 30 \quad T = 34 \]
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 $\frac{3}{2}$-approximation algorithm

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

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

$T \leq \max$ processing time $\leq T^*$
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.

\[
\begin{array}{ccccccc}
\text{jobs processing times} & t_1 & t_2 & t_3 & \ldots & t_m & t_{m+1} \\
\rightarrow & t_1 & t_2 & t_3 & \ldots & t_m & t_{m+1} \\
\end{array}
\]

Last job assigned \( \leq t_{m+1} \)

\[ T^* \geq 2 \cdot t_{m+1} \Rightarrow \text{last job assigned} \leq t_{m+1} \leq \frac{T^*}{2} \]

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

\[ T = \text{last job assigned} + \text{total load before the last job} \leq \frac{T^*}{2} + \frac{T^*}{2} = \frac{3T^*}{2} \]

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 \( O(n^{1/\varepsilon^2}) \).